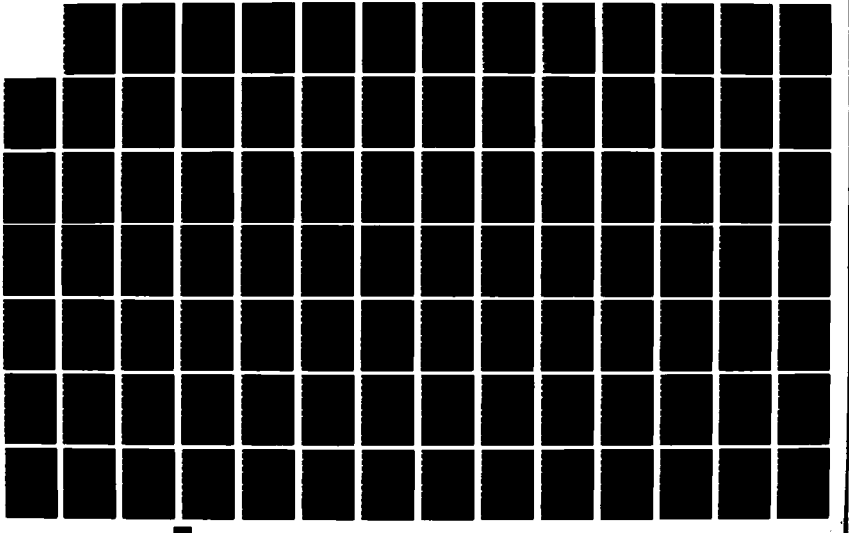
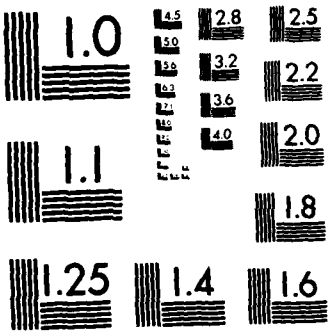
AD-A171 294 THE CLASSIFICATION OF MULTI-EDGE SHAPES USING AN AUTOREGRESSIVE MODEL AND (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH R D KENNETT DEC 85 1/2

UNCLASSIFIED AFIT/CI/NR-86-132T F/G 6/4 NL





SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFIT/CI/NR 86- 132T
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE *(and Subtitle)*: The Classification Of Multi-Edge Shapes Using An Autoregressive Model And The Karhunen-Loeve Expansion
5. TYPE OF REPORT & PERIOD COVERED: THESIS/DISSERTATION
6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): Ruth DeBeurs Kennett
8. CONTRACT OR GRANT NUMBER(s):

9. PERFORMING ORGANIZATION NAME AND ADDRESS: AFIT STUDENT AT: University of New Hampshire
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:

11. CONTROLLING OFFICE NAME AND ADDRESS: AFIT/NR, WPAFB OH 45433-6583
12. REPORT DATE: 1985
13. NUMBER OF PAGES: 136

14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*:
15. SECURITY CLASS. *(of this report)*: UNCLAS
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT *(of this Report)*: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AUG 26 1986

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*:

18. SUPPLEMENTARY NOTES: APPROVED FOR PUBLIC RELEASE: IAW AFR 190-1

LYNN E. WOLAVER 13 Aug 86
Dean for Research and Professional Development
AFIT/NR

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*:

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*: ATTACHED.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

86 8 28 026

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

THE CLASSIFICATION OF MULTI-EDGE SHAPES USING AN AUTOREGRESSIVE MODEL AND THE KARHUNEN-LOEVE EXPANSION

BY

RUTH DEBEURS KENNETT

B.S.E.E., University of New Hampshire, 1983

A THESIS

Submitted to the University of New Hampshire
in Partial Fulfillment of
the Requirements for the Degree of

Master of Science
in
Electrical Engineering

December, 1985

This thesis has been examined and approved.

Filson H. Glanz

Thesis director, Filson H. Glanz
Associate Professor of Electrical and
Computer Engineering

Paul J. Nahin

Paul J. Nahin, Associate Professor of
Electrical and Computer Engineering

L Gordon Kraft

Gordon L. Kraft, Assistant Professor of
Electrical and Computer Engineering

8-8-85

Date

ACKNOWLEDGEMENTS

I thank Dr. Filson H. Glanz for his excellent guidance and thoughtful encouragement during the progress of this thesis. I also thank my family, Crosby, Rachel, and Raymond for their unselfish support of this and all my previous academic endeavors.

I also wish to thank the following special individuals: Carl Piel for his expert knowledge of the VAX system, Aaron Pailes for his helpful suggestions and Nan Collins for her beautiful typing.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

THE CLASSIFICATION OF MULTI-EDGE SHAPES USING AN AUTOREGRESSIVE MODEL AND THE KARHUNEN-LOEVE EXPANSION

by

Ruth DeBeurs Kennett
University of New Hampshire, December, 1985

In this thesis a pattern recognition system capable of classifying two dimensional shapes with multiple edges was developed. The problem of multiple edge classification was treated as an extension of the single edge problem. For each edge, a feature vector was formed from the parameters of an autoregressive model of a time series representing the shape of the edge. The dimension of these feature vectors was further reduced by the use of a transformation based on the Karhunen-Loeve expansion. A minimum distance classification rule was used to classify an input transformed feature vector according to the nearest class mean in the transformed feature space.

Two boundary sampling methods as well as two versions of the Karhunen-Loeve transformation were investigated. An illustrative numerical example and the description of the system tests are provided. Using an equal angle boundary sampling technique and the pre-whitened Karhunen-Loeve transformation, an industrial shapes test showed 100% correct classification results with an average classification time of 1.27 seconds. The complete Fortran listings of the routines written for this system are included in the Appendix at the back of this work.

CHAPTER 1

INTRODUCTION

The purpose of this thesis is to classify objects with multiple edges using the method of classification based on a Karhunen-Loeve transformation of the autoregressive model parameters which represent the shapes of the boundaries detected in thresholded, digital images of the objects.

The human ability to recognize a particular object can be broken down into a series of logical steps. For instance in order to recognize a cat, a person looks at the cat and unconsciously collects the unique discriminating features of the animal. Then a comparing process occurs where the just collected information is compared to the vast storehouse of labelled features. A match occurs when the newly collected features agree with the features remembered as those belonging to a cat.

An automatic pattern recognition system uses the same principles. The block diagram of a simple pattern recognition system is shown below in Figure 1.1.

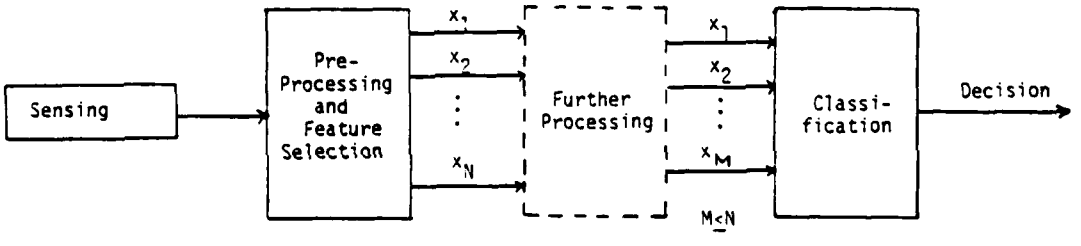


Figure 1.1.1 Block Diagram of a Pattern Recognition System

A sensing device such as a TV camera passes the raw data to a storage device. In the storage area the two-dimensional image can be digitized, thresholded, segmented and preprocessed in any way to enhance the discriminating features of the object. Once the image has been satisfactorily preprocessed, the physical features or the structural relationships (or both) can be selected and put into array (also called pattern vector or feature vector) form for computer manipulation. At this point the pattern vector can undergo further processing, or it can go directly to a classifier or categorizor. Generally the classifier has been previously trained with examples of expected pattern vectors called training samples and their corresponding labels. Pattern classification occurs when the input data is assigned to one of a finite number of categories or pattern classes.

In this thesis a pattern recognition system which is capable of classifying a broad set of objects is developed. The feature vectors of the objects are formed from the parameters of a particular linear difference equation called the autoregressive (AR) equation. The AR equation has been shown to be an effective model for the shape of an object's boundary [Dubois, 1984]. To further reduce storage space and classification time, the AR parameters are then projected onto the Karhunen-Loeve (K-L) coordinate axes. By making use of the special properties of the K-L expansion, the dimension of the feature vector can be further reduced. The result of this work has been the development of an efficient object recognition system that requires very little training data storage space and no extra dedicated hardware.

The list of applications for an automatic pattern recognition system is almost endless. At present, recognition systems are being used in

industry to inspect and identify parts. As a part moves down an assembly line the vision system can inspect for quality and can also make sure that the right part is moving down the line at the proper time. Electronic suppliers use vision systems to ensure proper placement of circuit board components [Beavers and Hubach, 1984]. Industrial robots are currently in use for parts inspection, sorting, and material handling. Two specialized bin-picking robots are already finding their way into factories; one is the GE Bin Vision robot and the other is the ORS (Object Recognition System) system called i-bot 1 [Edson, 1984]. Much research is being done for the autonomous robot where the ability to recognize obstacles is necessary to prevent the robot from walking into things or falling off precipices.

Other areas for the application of a pattern recognition system are in the military and the medical fields. A recognition system such as the one developed in this work could be adapted for use in the classification of objects in remotely sensed images. Other military applications for a pattern recognition system include automatic undersea searching and target classification. In the medical field this work could be implemented to classify cells or chromosomes.

The vision systems in use today generally require a great amount of training data in order to classify an object. A reduction in the amount of data needed to characterize an object would allow more storage space for the training data of additional objects. This would allow the vision system to be able to recognize more objects and less time would be needed for retraining. The recognition of three dimensional shapes may require the storage of many views of the same object. This is only possible if the amount of data to describe one view is minimal. The development of

a recognition system that could classify an object using a few parameters would be beneficial to any system that needed to classify a large number of objects.

The subsequent chapters will introduce and then describe in detail the theory and implementation of the pattern recognition system developed in this work. Chapter 2 introduces some feature selection techniques found in the literature. The rest of the chapter then focuses on the particular autoregressive model and the parameter estimation algorithm used to produce a feature vector. Chapter 3 describes the classification process and how that process is made simpler and quicker by transforming the feature vector onto a set of Karhunen-Loeve coordinate axes. Chapter 4 discusses the actual software implementation of the pattern recognition theory in order to produce a working system. Chapter 5 provides a numerical example of the system theories and also covers the tests of the system and their results. Chapter 6 concludes this work with some comments on the test results and suggestions for further improvements.

CHAPTER 2

FEATURE SELECTION THEORY

According to George S. Sebestyen "Pattern recognition is a process of decision making in which a new input is recognized as a member of a given class by comparison of its attributes with the already known pattern of common attributes of members of that class" [Sebestyen, 1962]. It is the job of the feature selector to form the set of attributes which are most representative of the common features of a class.

As a way of introducing the problems encountered in the feature selection process, some algorithms found in the literature will be described and compared in the first section. The second section will present the feature selector used in this work. Included in that section will be the description of some boundary representation schemes, and also the theory and parameter estimation of the model used to produce the feature vector.

2.1 Feature Selection Background

Feature selection algorithms are divided into three basic groups. The first group is comprised of heuristic algorithms which are based on ad hoc rules dependent on the particular objects to be classified. Since we are not dealing with any one particular type of object, we are more interested in general algorithms which can be applied to any type of object. These algorithms form the following two categories: the syntactical and the mathematical. The main difference between the two

types of algorithms is that the syntactical algorithms deal with the structural information of the image. This differs from the mathematical approach which fits an analytic model to the physical features of an image. These two types of algorithms are described in further detail in the following paragraphs.

2.1.1 Syntactical Approach to Feature Selection

Just as the meaning of a word is often context dependent so is the shape of an object's feature. Thus it can be assumed that the handle of a cup will be curved and will probably join the cup at two contact points. The syntactic or linguistic approach decomposes complex patterns recursively into simpler subpatterns in the same way that a sentence can be decomposed into letters. Patterns are described by their basic elements or subpatterns along with a set of syntactic rules or pattern grammar. In the classification of patterns described by a syntactic algorithm, the classifier performs a syntax analysis while parsing the pattern and answers the question of whether or not the pattern belongs to the language generated by the grammar. As can be deduced, this method is very complex to implement due to the very broad range of objects it tries to classify. Much research is being done in this area.

2.1.2 Mathematical Approach to Feature Selection

One of the earliest mathematical feature selection techniques is the method of moments. In this method a pattern is represented by its two dimensional moments calculated from a density distribution function with respect to a pair of axes fixed in the visual field [Hu, 1962]. These moments and other moments found in a similar manner can be formed into

linear combinations called "moment invariants." These are so-called since they are invariant under a number of similarity transformations. The major disadvantage of this method is that although the first few moments convey significant information for simple objects, they fail to do so for more complicated ones. Furthermore the computational requirements are substantial [Pavlidis, 1978].

Another mathematical feature selection technique is based on the values of the Fourier Transform of some characteristic of the pattern. This technique is the basis of algorithms which involve the calculation of the Discrete Fourier Transform (DFT) of an object's characterizing function which is commonly the boundary. The boundary samples can be expressed in terms of tangent angle vs arc length. They can also be expressed as terms of the complex function formed from the boundary sample position where the x axis of the reference plane corresponds to the real part of the sample and the y axis denotes the imaginary part. The main disadvantage of the resultant Fourier coefficient shape descriptors is that not all of the coefficients are invariant to translation and rotation [Granlund, 1972].

In the next section we will describe the theory of a different mathematical technique for feature selection based on the autoregressive equation. In that section we will describe how the AR model parameters can be estimated and how they are formed into a feature vector.

2.2 The Autoregressive Model

The autoregressive model is a probability model or stochastic model of an observed time series. It has been applied in the areas of spectral

analysis, speech recognition and transmission, and economic forecasting. In this section we will show how the autoregressive (AR) model can be used in shape recognition.

2.2.1 Boundary Representation Schemes

It is our goal to compress as many of the discriminating features of an object boundary(s) into as few AR parameters as possible. This goal emphasizes the need to accurately sample the boundary to catch the discriminating features. However we also need the resultant real valued time series to maintain the same form despite changes in the object's orientation, size, and initial sample point. If the object's centroid is known, there arise many possible methods of sampling the boundary to form the desired time series. These methods will be described below.

2.2.1.1 Equi-Angle Sampling. Consider that the object centroid is the origin of the cartesian system. Then N radius vectors can be projected from the origin to the boundary of the object as shown below.

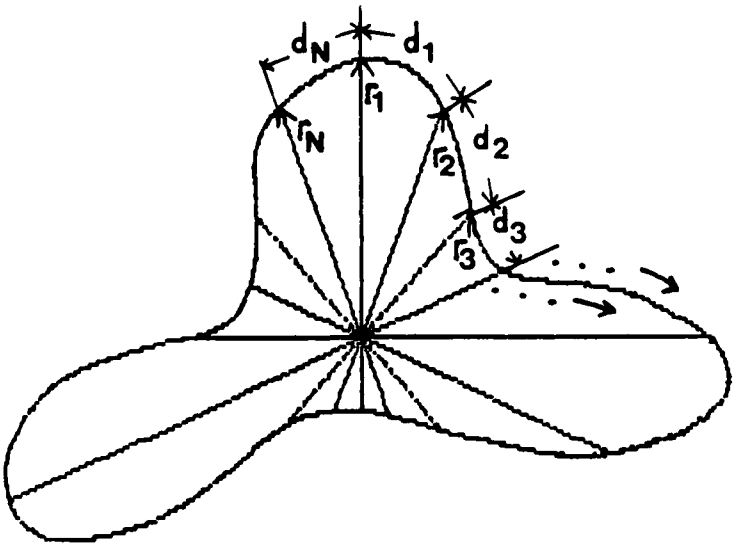


Figure 2.2.1 Example of Equal Angle Boundary Sampling Method

Let us make the requirement that the angle between consecutive radius vectors is constant and equal to $2\pi/N$. As Figure 2.2.1 shows, there are two possible time series produced from this type of representation. The first is the time series $r_1, r_2, r_3, \ldots, r_N$ formed from the lengths of the radius vectors. The second time series is $d_1, d_2, d_3, \ldots, d_N$ formed from the distances along the boundary between consecutive radius vector to boundary intersection points. Using this equal-angle sampling method, the portions of the boundary closer to the centroid are sampled relatively more frequently than the portions farther away from the centroid. Also in the case of highly curved shapes, a large segment of the boundary may not be sampled at all. For example in Figure 2.2.2 the tips of the letter S aren't sampled.

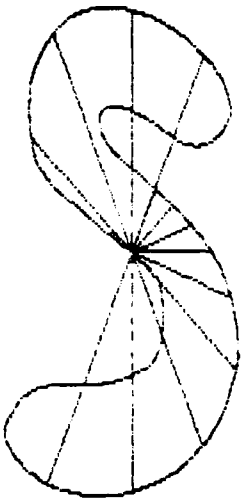


Figure 2.2.2 Example of Boundary Sampling Method Showing How Sections of Boundary Can Be Skipped

This makes the letter look cut off at the ends and the time series representation of this will differ from the one when the letter is rotated so that the tip is sampled.

A way to eliminate these problems is to sample the boundary so that one sample is always the same distance away from another sample. This method is described in the next section.

2.2.1.2 Equal-Arc Length Sampling. As before, consider the origin of the Cartesian coordinate system to be the centroid of the object boundary. Again we project N radius vectors from the centroid to the boundary. However, this time we make the requirement that the arc-length between boundary intersctions remain the same and proportional to the total length of the boundary. This boundary representation is shown below in Figure 2.2.2.

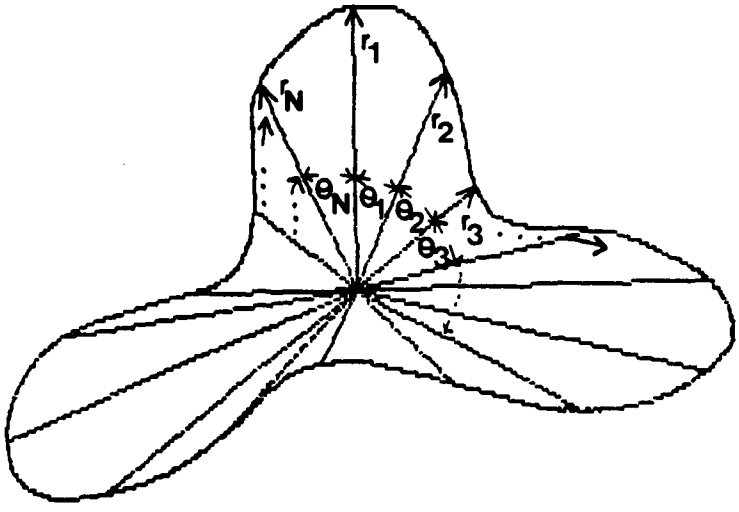


Figure 2.2.3 Example of Equal Arc Length Boundary Sampling Method

As can be seen, the boundary is now uniformly sampled. As before, we can form two time series from this method of boundary sampling. The

first is the time series $r_1, r_2, r_3, \ldots, r_N$ formed from the radius vector lengths. The second is the series $\theta_1, \theta_2, \ldots, \theta_N$ formed from the angles between $r_1, r_2, \ldots, r_N$.

Out of the four methods, two were actually implemented. Both implementations formed a time series from the radius vector lengths. One time series was formed using the equal angle sampling method and the other time series was formed using the equal arc length sampling method. Let us make a requirement that the boundary be closed and that it may not cross over itself. We then have a discrete time series that is stationary and periodic.

2.2.2 The General Autoregressive Model

In the general AR model the current value of the process is expressed as a finite, linear aggregate of previous values of the process and a zero mean white noise term e_t [Box and Jenkins, 1976]. If we let the values of the process at equally spaced time intervals t, t-1, t-2, ... be y_t, y_{t-1}, y_{t-2}, ... then

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + e_t \qquad (2.2.1)$$

This is called an autoregressive process of order p. In Equation 2.2.1 the present value of variable y is regressed on the previous values of itself. Thus the model is autoregressive. The above equation can be written more compactly as

$$y_t = \sum_{i=1}^{p} \phi_i y_{t-i} + e_t \qquad (2.2.2)$$

The model contains p+1 unknown parameters ϕ_1, ϕ_2, ϕ_p , and σ^2 which have to be estimated from the data. The additional parameter σ^2 is the variance of the white noise process e_t.

In the frequency domain the AR model can be formed into a transfer function which has no zeros. Therefore the AR model is also called the all-pole model. The AR model is also linked with the maximum entropy method (MEM) in the area of spectral analysis.

2.2.3 The Specific Autoregressive Model

As shown by Dubois [Dubois, 1984] an effective AR model for use in shape recognition is:

$$r_t = \alpha + \sum_{j=1}^{m} \theta_j r_{t-j} + \sqrt{\beta}\omega_t \qquad (2.2.3)$$

whose form was originally suggested by Kashyap and Chellappa [Kashyap and Chellapa, 1981]. In this form we have:

r_t = current radius vector length

r_{t-j} = the radius vector length detected j time intervals before the current r_t. Collectively these are called the lag terms.

m = model order

$\sqrt{\beta}\omega_t$ = current error

$\{\omega_t\}$ = a sequence of random, independent, zero mean samples (white noise sequence) which has unit variance, i.e. $E(\omega_i\omega_j) = 1$ if i=j and = 0 otherwise

$\theta_1,\ldots,\theta_m$ = the AR parameters or lag coefficients to be estimated from the time series

$\{\alpha,\beta\}$ = unknown constants to be estimated.

Looking back at the general autoregressive model of Equation 2.2.1 we see that the specific model used for shape recognition contains two additional terms α and $\sqrt{\beta}$. Comparing the two equations, $\sqrt{\beta}\omega_t$ corresponds to the e_t term of the general model.

2.2.3.1 Estimating α and β of the Specific AR Model. Let us assume that we already know the θ_i, $i = 1, \ldots, m$. Then we still need to find estimates of α and β in Equation 2.2.3. Now we will define the average radius vector length as $\bar{r}$ so that

$$\bar{r} = \frac{1}{N} \sum_{t=1}^{N} r_t \tag{2.2.4}$$

where N is the number of radius vectors in the object boundary representation. We can rewrite the general form of Equation 2.2.2 to model the set of differences $r_t - \bar{r}$ as:

$$r_t - \bar{r} = \sum_{j=1}^{m} \theta_j (r_{t-j} - \bar{r}) + \sqrt{\beta}\omega_t \tag{2.2.5}$$

or

$$r_t = \sum_{j=1}^{m} \theta_j r_{t-j} + \sqrt{\beta}\omega_t + \bar{r} - \sum_{j=1}^{m} \theta_j \bar{r}$$

Substituting in Equation 2.2.2 for r_t and cancelling the common terms we find the constant α can be estimated by the equation:

$$\alpha = \bar{r} \left(1 - \sum_{j=1}^{m} \theta_j\right) \tag{2.2.6}$$

Note that α is directly proportional to the mean radius vector length and thus is an indicator of a shape's size.

Now that we have an estimate for α we still need to find an estimate of β. Rewriting Equation 2.2.3 to separate the term we have

$$\sqrt{\beta}\omega_t = r_t - \alpha - \sum_{j=1}^{m} \theta_j r_{t-j} \tag{2.2.7}$$

Squaring both sides and taking the expectation of the results we get

$$\beta E[\omega_t^2] = E\left[\left(r_t - \alpha - \sum_{j=1}^{m} \theta_j r_{t-j}\right)^2\right]$$

Since the sequence of ω_t has unit variance we call β the residual variance and it can be estimated as:

$$\beta = \frac{1}{N} \sum_{t=1}^{N} \left(r_t - \alpha - \sum_{j=1}^{m} \theta_j r_{t-j}\right)^2 \tag{2.2.8}$$

Looking at Equation 2.2.8 we see that β is roughly proportional to the average of the squared radius vector lengths. We may conclude then that a possible size invariant shape descriptor would be in the form $\alpha/\sqrt{\beta}$.

2.2.3.2 Estimating the Coefficients of the Specific AR Model It has been shown in the thesis of Dubois [Dubois, 1984] that the AR parameters $\vartheta_1, \ldots, \vartheta_m, \alpha$ can be found by using the method of least squares. In that work the model parameters were chosen so as to minimize the expected value of the squared error term β in Equation 2.2.8. In the experimental work of Dubois the equations were solved using matrix methods which required $(m+1)^3/3$ plus on the order of $(m+1)^2$ operations and $(m+1)^2$ storage locations [Makhoul, 1975].

Due to the special properties of the AR equations in terms of the auto-correlation function the parameters of the model can be estimated recursively.

We use the general form of the AR equation which models the set of differences $r_t-\bar{r}$ in Equation 2.2.5. With the α term as defined in Equation 2.2.6 we find that the general AR model of Equation 2.2.2 based on these differences, is equivalent to the specific AR model of Equation 2.2.3 suggested by Kashyap and Chellappa.

If we multiply both sides of Equation 2.2.5 by the term $(r_{t-k}-\bar{r})$ we get [Box and Jenkins, 1976]:

$$(r_{t-k}-\bar{r})(r_t-\bar{r}) = \sum_{j=1}^{m} \vartheta_j (r_{t-k}-\bar{r})(r_{t-j}-\bar{r}) + (r_{t-k}-\bar{r})\sqrt{\beta}\omega_t$$

Finding the expectation of both sides leads to:

$$E[(r_{t-k}-\bar{r})(r_{t-r}-\bar{r})] = E[\sum_{j=1}^{m} \vartheta_j (r_{t-k}-\bar{r})(r_{t-j}-\bar{r})] + E[(r_{t-k}-\bar{r})\sqrt{\beta}\omega_t]$$

$$k = 0,\ldots,m \qquad (2.2.9)$$

The term $E[(r_{t-k}-\bar{r})\sqrt{\beta}\omega_t]$ vanishes for k>0 since the sequence ω_t is uncorrelated with the $(r_{t-k}-\bar{r})$ term. For k=0 this term is $E[\sum_{j=1}^{m} \theta_j r_{t-j}\sqrt{\beta}\omega_t + \beta\omega_t^2]$ which is equal to $\beta E[\omega_t^2]$ and is defined as σ_m^2. Let us define

$$R_{k-j} = E[(r_{t-k}-\bar{r})(r_{t-j}-\bar{r})] \cong \frac{1}{N}\sum_{t=1}^{N}(r_{t-k}-\bar{r})(r_{t-j}-\bar{r})$$

which is the covariance function. Note that $R_{-i} = R_i$. Rewriting Equation 2.2.9 and changing the order of summations on the right side we get:

$$R_k = \theta_1 R_{k-1} + \theta_2 R_{k-2} + \ldots + \theta_m R_{k-m} \text{ for } k>0 \qquad (2.2.10)$$

We now have the autoregressive difference equation in terms of the autocorrelation function of the sampled data.

Equation 2.2.10 applies for any model order of the AR process. Since there is a different set of parameters for each model order we will denote the kth coefficient in an AR process of order m as θ_{mk}. The last coefficient at k = m is thus θ_{mm}. This term is known as the partial autocorrelation coefficient and also is referred to as the mth reflection coefficient. We can now form a set of equations, one for each model order.

$$R_k = \theta_{m1} R_{k-1} + \theta_{m2} R_{k-2} + \ldots + \theta_{m,n-1} R_{k-m+1} + \theta_{mm} R_{k-m}$$
$$\text{for } k = 1, \ldots, m \qquad (2.2.11)$$

These are called the Yule-Walker equations. In matrix form these equations may be written

$$\begin{bmatrix} R_o & R_1 & R_2 & \cdots & R_{m-1} \\ R_1 & R_o & R_1 & \cdots & R_{m-2} \\ R_2 & R_1 & R_o & \cdots & R_{m-3} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{m-1} & R_{m-2} & R_{m-3} & \cdots & R_o \end{bmatrix} \begin{bmatrix} \theta_{m1} \\ \theta_{m2} \\ \theta_{m3} \\ \vdots \\ \theta_{mm} \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ \vdots \\ R_m \end{bmatrix} \qquad (2.2.12)$$

The above equations form an mxm autocorrelation matrix which is Toeplitz

since all the elements along the diagonal are identical. Also the matrix is positive definite which follows from the positive definite property of the autocorrelation function.

If all the terms in the mxm matrix of Equation 2.2.12 are normalized to R_o, the correlation lag at t = 0, the Yule-Walker equations can be written in terms of the resultant normalized autocorrelation coefficients [Makhoul, 1975]. Thus we get

$$\begin{bmatrix} 1 & C_1 & C_2 & \cdots & C_{m-1} \\ C_1 & 1 & C_1 & \cdots & C_{m-2} \\ C_2 & C_1 & 1 & \cdots & C_{m-3} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{m-1} & C_{m-2} & C_{m-3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} \theta_{m1} \\ \theta_{m2} \\ \theta_{m3} \\ \vdots \\ \theta_{mm} \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ \vdots \\ C_m \end{bmatrix} \qquad (2.2.13)$$

where $C_i = \frac{R_i}{R_o}$ $\quad i = 0,\ldots,m$

By recognizing that the right hand column vector contains mostly the same elements as in the autocorrelation matrix, the equations can be solved using simple algebraic reduction techniques as shown in Box and Jenkins [Box and Jenkins, 1976].

The solution of the equations reduces to the following recurrence equations:

$$\theta_{mm} = \frac{C_m - \sum_{j=1}^{m-1} \theta_{m-1,j} C_{m-j}}{1 - \sum_{j=1}^{m-1} \theta_{m-1,j} C_j} \qquad j=1, \ldots, m-1 \qquad (2.2.14a)$$

$$\theta_{mj} = \theta_{m-1,j} - \theta_{mm} \theta_{m-1,m-j} \qquad (2.2.14b)$$

These equations are attributed to Durbin [Durbin, 1960]. They show how the parameters at the desired model order are calculated using the parameters of all the previous model orders. To find the mth order process parameter set, the parameter sets $\{\theta_{11}\}$, $\{\theta_{21},\theta_{22}\},\ldots,$ $\{\theta_{m1},\theta_{m2},\ldots,$ $\theta_{mm}\}$ are all calculated. This recursive method requires only 2m storage locations and m^2 plus on the order of m operations [Makhoul, 1975]. We can now augment Equation 2.2.12 which represent the k=1,...,m equations of 2.2.9 with the k=0 equation: $R_o = \sum_{j=1}^{m} \theta_j R_j + \sigma_m^2$. The result becomes:

$$\begin{bmatrix} \bar{R}_o & R_1 & R_2 & \cdots & R_m \\ R_1 & R_o & R_1 & \cdots & R_{m-1} \\ R_2 & R_1 & R_o & \cdots & R_{m-2} \\ \vdots & \vdots & \vdots & & \vdots \\ R_m & R_{m-1} & R_{m-2} & \cdots & R_o \end{bmatrix} \begin{bmatrix} \bar{1} \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_m \end{bmatrix} = \begin{bmatrix} \bar{\sigma}_m^{\bar{2}} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

To solve the above equations we use a set of recurrence relations which produce the same θ_is as the Durbin algorithm, except for a sign change on the parameters due to assuming the general AR model is $\sum_{j=0}^{m} \theta_j y_{t-j} = e_t$. The algorithm is initialized by:

$$\theta_{11} = -R_1/R_o \qquad (2.2.15a)$$

$$\sigma_1^2 = (1-\theta_{11}^2)R_o \qquad (2.2.15b)$$

the recursion for k = 2,3,...,m is given by

$$\theta_{kk} = -[R_k + \sum_{j=1}^{k-1} \theta_{k-1,j} R_{k-j}]/\sigma^2_{k-1} \qquad (2.2.15c)$$

$$\theta_{ki} = \theta_{k-1,i} + \theta_{kk}\theta_{k-1,k-i} \qquad (2.2.15d)$$

$$\sigma^2_k = (1 - \theta_{kk}^2)\sigma^2_{k-1} \qquad (2.2.15e)$$

This form of the recurrence equations has two advantages over the original equations 2.2.14. First, the correlation coefficients need not be normalized before being passed to the algorithm. Also the σ_k^2 term can be used in the model order selection process. The correct model order is chosen when the AR model has arrived at the best fit to the observed time series.

If the model order is chosen too high or too low, the AR model will not provide an accurate representation of the sampled process. In the σ_k^2 equation (Equation 2.2.15c) we see that σ_k^2 is dependent solely on the present and past values of the partial correlation coefficient. The term is thus an indicator of the error and has the property that it decreases (or remains the same) as the order of the model increases [Makhoul, 1975]. A suitable model order for the process can be chosen when the change in the error term i.e., $\sigma_k^2 - \sigma^2_{k-1}$ is below a specified tolerance level.

2.2.4 The Autoregressive Model Parameters as Shape Descriptors

The shape descriptors form a feature vector which is a string or set of numbers which represent the features or characteristics of the shape. Since the physical features of a shape are invariant to rotation, translation, and scale, so should the numerical shape descriptors be invariant to rotation, translation and scale. Also the numerical shape descriptors should be invariant to where the sampling process begins. From the previous section we have estimated the following AR parameters: θ_i, $i = 1,\ldots,m$ estimated using the recursive algorithm of Equation 2.2.15, α as in Equation 2.2.6 and β from Equation 2.2.8. These parameters

in vector form $[\theta_1, \theta_2, \ldots, \theta_m, \alpha/\sqrt{\beta}]^t$ produce a feature vector which has the desired invariant properties [Dubois, 1984]. However, it should be noted that, in using the equi-angle boundary sampling method, the rotation and starting point invariance property only applies when the change in rotation or starting point is a multiple of $2\pi/N$. If N is sufficiently large, this no longer is a problem.

CHAPTER 3

CLASSIFICATION OF THE TRANSFORMED FEATURE VECTOR

Using the theory from Chapter 2, it is possible to form a feature vector from the set of AR parameters which describe the shape of an object's boundary. If we had a bin full of the exact same object and if we found the feature vector of the outer boundary of each object using the same method, we would find that most feature vectors would be slightly different from the rest. This is due not only to minor variations in rotation, i.e. not exact multiples of $2\pi/N$, but also to pixel quantization error and round-off errors in the system. Considered as a class however, all these feature vectors will be roughly the same. It is then possible to use a classification scheme based on the feature vector of the samples of each class. Some of these classification schemes as found in the literature will be described in the next section. It will become apparent that there is usually a need for further processing of the feature vector in order to correctly classify the shape of the object's boundary. This is especially true for objects with very similar shape. Thus the Karhunen-Loeve (K-L) expansion will be introduced and two implementations of the K-L expansion will be described. The last section of this chapter will be devoted to the particular classification rule used in the experimental work of this thesis.

3.1 Background of Classification Theory

The pattern classification problem is essentially the problem of partitioning the feature space so as to assign each possible feature

vector or point in that space to the proper pattern class. The classification techniques in the literature form three groups depending on how much a priori information about the feature vector to be classified is known. The first group comprises the deterministic techniques wherein no assumptions are made on the statistical properties of a pattern class. When the statistical properties of the feature vector are known, then it is possible to use the group of statistical classification techniques. A third group which allows incomplete knowledge of the statistical properties of the feature vectors is the group of trainable classifiers. These three methods of classification will be described in further detail in the next pages.

3.1.1 Deterministic Classification Techniques

This group of classification techniques depends on the formulation of mathematical, deterministic, discriminant functions which are used to partition the feature space. These functions assume that the feature vector components are deterministic quantities. The hyperplane classification method, the third technique used in the thesis of Dubois [Dubois, 1984], is an implementation of a deterministic discriminant function.

The general form of a discriminant function is as follows [Fu, 1968]. Let $\omega_1, \omega_2, \ldots, \omega_c$ be designated as the c possible pattern classes to be recognized and let

$$\underline{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_d \end{bmatrix}$$

be the feature vector where y_i represents the ith feature. The discriminant function $D_j(Y)$ associated with pattern class ω_j, j=1,..., c is such that if the feature vector $\underline{Y}$ is in class ω_i then the value of $D_i(Y)$ must be the largest (or possibly the smallest) of all the $D_j(Y)$ for $j \neq i$. The discriminant functions form decision boundaries between regions associated with class ω_i and ω_j which can be expressed as:

$$D_i(Y) - D_j(Y) = 0$$

An example of the discriminant function for a two class problem is shown in Figure 3.1.1.

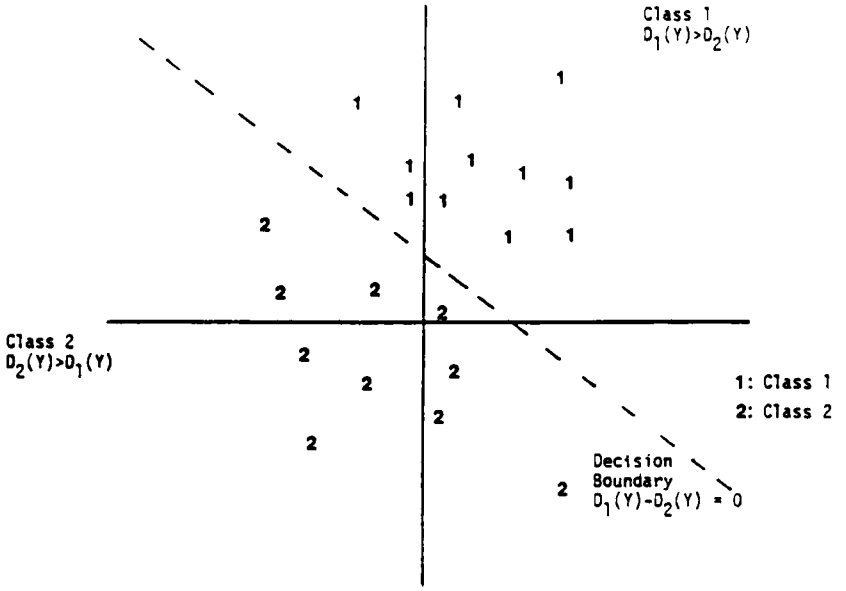


Figure 3.1.1 Example of a Discriminant Function for a 2 Class Problem

An important class of linear deterministic classifiers uses as the classification criterion the distance between the unlabelled feature point and the labelled points of the feature space. A discriminant function for such a so called <u>minimum distance classifier</u> is:

$$D_j(Y) = \sum_{n=1}^{N} \left| \sum_{i=1}^{\bar{d}} (Y_i - R_{j,n,i})^2 \right|^{1/2} \qquad j = 1,\ldots,c \tag{3.1.1}$$

where N is the number of samples of the jth class and $R_{j,n,i}$ is the ith component of the nth sample of the jth class. Equation 3.1.1 can be implemented more simply without loss of information as:

$$D_j(Y)^2 = \sum_{n=1}^{N} \sum_{i=1}^{d} (Y_i - R_{j,n,i})^2 \qquad j = 1,\ldots,c$$

Classification of the unlabelled feature vector $\underline{Y}$ is based on the smallest class distance $D_j(Y)$. The performance of the minimum distance classifier is dependent on whether or not the training sample points form tight clusters that are well separated from each other. In classification terminology we wish for the training samples to form clusters such that their intraset, or within-class, distances are small and their interset, or between-class, distances are large.

3.1.2 Statistical Classification Techniques

In the deterministic classification techniques, it is assumed that the feature vector components are deterministic quantities. However, in many cases the noise effect due to large variations in the feature vector components cannot be neglected. If we assume that for each class the probability density function of the feature vector is known and the probability of the ith class occurring is known, then the classification problem becomes a statistical problem. The task of the classifier is then to minimize the probability of misclassification [Fu, 1968].

The statistical approach to the pattern recognition problem is generally based on the Bayes rule which presents an optimum measure of classification performance [Tou and Gonzalez, 1974]. The classifier that uses the Bayes rule is called a Bayes classifier. One basic Bayes classifier is the implementation of the following discriminant function:

$$D_j(Y) = P(Y/\omega_j)p(\omega_j) \qquad j = 1,\ldots,c$$

A pattern vector $\underline{Y}$ is assigned to class ω_j if for that class $D_j(Y) > D_i(Y)$ for all $i \neq j$. The main problem with the statistical classification techniques is that of estimating the probability density function characterizing each of the different pattern classes. This implies that the computational requirements of these techniques will probably be greater than for the deterministic classification techniques.

3.1.3 Trainable Pattern Classifiers

The trainable pattern classifiers produce discriminant functions by means of iterative learning algorithms. These classifers can use either the deterministic or statistical techniques or a combination of both to produce the desired discriminant functions. The main use of a trainable classifier is when the required information for optimal pattern classification is only partially known. During its operation the classifier learns the needed information. If the learned information gradually approaches the true information, then the decisions based on the learned information will eventuallly approach the optimal decision as if all the information required is known [Fu, 1968]. The trainable classifiers are divided into two groups: the supervised classifiers and the unsupervised classifiers. The difference between the two groups is that the supervised classifiers have available the correct classification of the observed patterns while the unsupervised classifiers don't. Again, the computational demands of the trainable classifier algorithms are generally greater in comparison to the deterministic techniques.

3.2 The Karhunen-Loeve Expansion

It is our goal to classify objects as quickly and accurately as possible. Thus it is desired that the actual classifier should be a simple deterministic minimum distance classifier. As was stated previously, for the minimum-distance classifier to give accurate results, it is necessary for the feature space to form tight clusters well separated from each other. Also if the feature vector dimension can be reduced, the classification time will decrease. In this section we will introduce the Karhunen-Loeve expansion and its properties which allow us to achieve quick, accurate classification of feature vectors of reduced dimension. Two implementations of the K-L expansion will be described and finally the actual minimum-distance classifier used in this work will be introduced.

3.2.1 Basic Theory of Karhunen-Loeve Coordinate Axes

Let us, first of all, introduce the Karhunen-Loeve expansion. We know that a periodic function can be expanded into a Fourier series:

$$x(t) = \sum_{n=-\infty}^{\infty} x_n \exp(jn\omega_o t) \qquad (3.2.1)$$

where ω_o is the angular frequency and the x_n are the Fourier coefficients. A similar expansion can be formed for a non-periodic function x(t) where

$$x(t) = \sum_{n=1}^{\infty} y_n u_n(t) \qquad 0 \leq t \leq T \qquad (3.2.2)$$

Suppose we use only D discrete sampling points to approximate the time function during the time interval $0 \leq t \leq T$, then x(t) becomes the column vector $\underline{x}$. Also the generalized Karhunen-Loeve expansion of Equation 3.2.2 becomes

$$\underline{x} = \sum_{n=1}^{D} y_n \underline{u}_n \tag{3.2.3}$$

where the vectors $\underline{u}_n$ form a deterministic orthornormal basis set. The basis vector $\underline{u}_n$ is D-dimensional and represented as

$$\underline{u}_n = \begin{vmatrix} u_{n1} \\ u_{n2} \\ \vdots \\ u_{nD} \end{vmatrix} \tag{3.2.4}$$

The derivation of the Karhunen-Loeve coordinate axes from the K-L expansion is described next, following the method of Devijver and Kittler [Devijver and Kittler, 1982]. We start by making an approximation to $\underline{x}$ of Equation 3.2.3. Let's call this approximation $\hat{x}$, and let it have dimension d where d<D i.e.

$$\hat{x} = \sum_{n=1}^{d} y_n \underline{u}_n \tag{3.2.5}$$

The problem is now of finding the vectors $\underline{u}_n$ such that the mean square error (MSE) between $\underline{x}$ and its approximation $\hat{\underline{x}}$ is minimized. Let this error be called ε where

$$\varepsilon = E\{(\underline{x}-\hat{\underline{x}})^t(\underline{x}-\hat{\underline{x}})\} \tag{3.2.6}$$

and the expected value E{} is over all samples in a class. Substituting in the expansions for $\underline{x}$ and $\hat{\underline{x}}$ we get

$$\varepsilon = E\{(\sum_{n=d+1}^{D} y_n \underline{u}_n)^t(\sum_{n=d+1}^{D} y_n \underline{u}_n)\} \tag{3.2.7}$$

Due to the orthonormality property of the basis vectors we know that

$$\underline{u}_n^t \underline{u}_m = \delta_{nm} \tag{3.2.8}$$

where δ_{nm} is the Kronecker delta and equals 1 if n=m and 0 otherwise. The error of Equation 3.2.7 reduces to

$$\varepsilon = E\{ \sum_{n=d+1}^{D} y_n^2 \} \tag{3.2.9}$$

Since we still want to minimize the error in terms of the vectors $\underline{u}_n$, let us find y_n in terms of $\underline{u}_n$. Since n is a dummy variable we can rename it and rewrite Equation 3.2.3 as

$$\underline{x} = \sum_{m=1}^{D} y_m \underline{u}_m \tag{3.2.10}$$

We can now multiply both sides of Equation 3.2.10 by $\underline{u}_n^t$ to get

$$\underline{u}_n^t \underline{x} = \underline{u}_n^t \sum_{m=1}^{D} y_m \underline{u}_m \tag{3.2.11}$$

Using the orthonormality property of Equation 3.2.8 we find that the right side of Equation 3.2.11 will be zero except when m=n. Thus we get:

$$y_n = \underline{u}_n^t \underline{x} \qquad n = 1,\ldots,D \tag{3.2.12}$$

which can be substituted back into Equation 3.2.9 i.e.:

$$\varepsilon = E\{ \sum_{n=d+1}^{D} \underline{u}_n^t \underline{x}\, \underline{x}^t \underline{u}_n \} \tag{3.2.13}$$

Since the $\underline{u}_n$ basis vectors are deterministic the order of summation and expectation can be interchanged to get:

$$\varepsilon = \sum_{n=d+1}^{D} \underline{u}_n^t [E\{\underline{x}\, \underline{x}^t\}] \underline{u}_n \tag{3.2.14}$$

We can denote the expectation term as ψ where

$$\psi = E\{\underline{x}\, \underline{x}^t\} \tag{3.2.15}$$

then the MSE of the approximation becomes

$$\varepsilon = \sum_{n=d+1}^{D} \underline{u}_n^t \psi \underline{u}_n \tag{3.2.16}$$

Since we wish to minimize the error ε at the same time as maintaining the constraint in Equation 3.2.8, we need to find a form of the error equation where the desired constraint is explicit. To find this we use the method of Lagrange multipliers. First we can rewrite the constraint as:

$$\underline{u}_n^t \underline{u}_n - 1 = 0 \tag{3.2.17}$$

Multiplying both sides of Equation 3.2.17 by a constant λ_n doesn't change the equality. Thus Equation 3.2.16 can be written as a function of $\underline{u}_n$ as:

$$g(\underline{u}_n) = \sum_{n=d+1}^{D} \underline{u}_n^t \psi \underline{u}_n - \sum_{n=d+1}^{D} \lambda_n (\underline{u}_n^t \underline{u}_n - 1) \tag{3.2.18}$$

If the above equation is differentiated with respect to $\underline{u}_n$ and the result is set equal to zero we get the following condition:

$$(\psi - \lambda_n I)\underline{u}_n = 0 \tag{3.2.19}$$

Thus the optimal vectors $\underline{u}_n$ are the eigenvectors of the matrix ψ and the constants λ_n are the corresponding eigenvalues. Since ψ is a DxD symmetric matrix, it will have at most D linearly independent, orthogonal eigenvectors and D real eigenvalues [Lanczos, 1956].

We still need to find the d eigenvectors out of the set of D in order to form the approximation $\hat{\underline{x}}$ in Equation 3.2.5. From Equation 3.2.19 we know that $\psi \underline{u}_n = \lambda_n \underline{u}_n$. Substituting this into Equation 3.2.16 and using the orthonormality condition again, we get the error in terms of the eigenvalues of ψ, i.e.

$$\varepsilon = \sum_{n=d+1}^{D} \underline{u}_n^t \lambda \underline{u}_n = \sum_{n=d+1}^{D} \lambda_n \tag{3.2.20}$$

Here we can see that to minimize ε, the sum on the right should include the smallest eigenvalues of the matrix ψ. This implies that $\hat{x}$ should be formed from the eigenvectors corresponding to the d largest eigenvalues of ψ i.e.:

$$\hat{x} = \sum_{n=1}^{d} y_n \underline{u}_n \tag{3.2.20}$$

Where the eigenvalues λ_n corresponding to $\underline{u}_n$ satisfy

$$\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_d \geq \cdots \geq \lambda_D \tag{3.2.21}$$

The vectors $\underline{u}_n$ form what are called the Karhunen-Loeve coordinate axes.

It has just been shown that in using the eigenvectors corresponding to the largest eigenvalues of the matrix ψ, the mean square error between the original vector and its approximation is minimized. Also it can be shown [Devijver and Kittler, 1982] that the coefficients y_n of the expansion are uncorrelated. This suggests how the K-L coordinate axes can be used to reduce the dimension of the input vector by decorrelating the components and removing the axes which do not convey much information [Kittler, 1975].

3.2.2 Application of the Karhunen-Loeve Coordinate System to the Recognition Problem

Using the results of Chapter Two we can form an m+1 dimensional feature vector from the parameters of the AR model. Let's say we would like to transform this feature vector into a d dimensional vector where $d < m+1$. Then the K-L coordinate axes can be applied to produce the desired reduced dimension transformed vector $\underline{y}$, i.e.

$$\underline{y} = V^t \underline{x} \tag{3.2.22}$$

The vector $\underline{x}$ represents the original m+1 dimensional feature vector and V is a Dxd matrix whose columns are formed from the eigenvectors of what is referred to as a K-L coordinate generating matrix [Devijver and Kittler, 1982].

In a previous section we found that the Karhunen-Loeve coordinate axes were found by solving for the eigenvalues and corresponding eigenvectors of matrix ψ defined as:

$$\psi = E\{\underline{x}\ \underline{x}^t\} \tag{3.2.23}$$

We could use this definition in our pattern recognition problem, but we wouldn't be taking advantage of the fact that the input data will be a

mixture set containing elements of many classes ω_i, i = 1,...,c where c is the number of pattern classes. If we assume that the samples of each class are normally distributed about the class mean vector, then each class of samples can be completely described by its mean vector, $\underline{m}_i$ and its covariance matrix C_i [Tou and Gonzalez, 1974] where C_i is defined as:

$$C_i = E[(\underline{x}-\underline{m}_i)(\underline{x}-\underline{m}_i)^t] \qquad i = 1,\ldots,c$$

$$\cong \frac{1}{n_i} \sum_{k=1}^{n_i} [(\underline{x}-\underline{m}_i)(\underline{x}-\underline{m}_i)^t] \qquad i = 1,\ldots,c \qquad (3.2.24)$$

where n_i is the number of samples of the ith class and

$$\underline{m}_i = E[\underline{x}_i] \qquad i = 1,\ldots,c$$

$$\cong \frac{1}{n_i} \sum_{k=1}^{n_i} x_{ik} \qquad i = 1,\ldots,c \qquad (3.2.25)$$

Using this additional information we can form a <u>within class scatter matrix</u> S_w [Devijver and Kittler, 1982] which is the average of all the class covariance matrices i.e.:

$$S_w = \sum_{i=1}^{c} P_i C_i \qquad (3.2.26)$$

where P_i is the apriori probability of a sample of the mixture set being from the ith class. If a sample is equally likely to come from one class as any other then:

$$P_i = \frac{1}{c} \qquad (3.2.27)$$

and the matrix S_w becomes:

$$S_w = \frac{1}{c} \sum_{i=1}^{c} \frac{1}{n_i} \sum_{k=1}^{n_i} (\underline{x}_{ik} - \underline{m}_i)(\underline{x}_{ik} - \underline{m}_i)^t \qquad (3.2.28)$$

Hence the columns of the matrix U in Equation (3.2.22) can be formed from the eigenvectors of S_w ordered by the decreasing value of their corresponding eigenvalues.

It can be shown that the population entropy can be minimized by forming the matrix V from the eigenvectors of matrix S_w corresponding to the smallest eigenvalues [Devijver and Kittler, 1982]. A good measure of intraset dispersion is the entropy function given by:

$$H = -E[\log p(\underline{y})]$$

where $p(\underline{y})$ represents the class probability density function [Tou and Gonzalez, 1974].

It is noted that the within class scatter matrix S_w is symmetric and positive semi-definite, thus the eigenvalues will be real and greater than or equal to zero. The eigenvectors point in the direction of the principle axes of a hyperellipsoid whose shape is defined by the covariance matrix and the corresponding eigenvalues determine the length of these axes. In other words, the eigenvalues represent the variance of the data in the direction of the corresponding eigenvectors in the feature space.

3.3 The Optimal Karhunen-Loeve Coordinate System

There are many variations to the Karhunen-Loeve coordinate system in the literature. These variations are specially tailored to their particular type of pattern recognition problem and essentially differ in the choice of the K-L system generating matrix. These techniques are inferior to the next method to be described in the sense that they do not utilize the class mean information in an optimal manner [Kittler, 1975]. In this section the theory of the optimal Karhonen-Loeve coordinate axes will be described.

Generally the most important discriminating information is contained in the class mean vectors $\underline{m}_i$. Let us then introduce a new matrix based on the class mean variances referred to as the between class scatter matrix S_b [Devijver and Kittler, 1982] where:

$$S_b = \sum_{i=1}^{c} P_i(\underline{m}_i - \bar{\underline{m}})(\underline{m}_i - \bar{\underline{m}})^t \qquad (3.3.1)$$

Here $\underline{m}_i$ is the class mean vector as defined in Equation 3.2.25 and $\bar{m}$ is the average of the class mean vectors i.e.:

$$\bar{m} = \frac{1}{c} \sum_{i=1}^{c} \underline{m}_i \qquad (3.3.2)$$

When there are c classes, the space spanned by the class mean vectors will be at most c-1 dimensional [Devijver and Kittler, 1982]. Here we see that by using matrix S_b as the K-L coordinate system generating matrix we may achieve a more efficient way of reducing the dimension of the pattern vector.

The algorithm as suggested by Kittler in [Devijver and Kittler, 1982] is described next. We first set out to decorrelate the noise on the pattern vectors. From the previous section we found that this is done by the transformation matrix V in Equation 3.2.22 formed from the eigenvectors of S_w. We then would like to normalize the variances of the noise components of the pattern vectors to unity. To do this we form a matrix Λ whose diagonal elements are the eigenvalues corresponding to the eigenvectors of V. Combining the two matrices we achieve the desired results by using the transformation matrix B where

$$B = V \Lambda^{-1/2} \qquad (3.3.3)$$

on the matrix S_w i.e.

$$B^t S_w B = I \qquad (3.3.4)$$

Similarly we can decorrelate and prewhiten the between class scatter matrix S_b of Equation 3.3.1 to form S_b' i.e.

$$S_b' = B^t S_b B \tag{3.3.5}$$

We find the optimal coordinate system by finding the eigenvalues and eigenvectors of the matrix S_b'. The optimal feature extracting transformation matrix is now W where

$$W = U\Lambda^{-1/2}V \tag{3.3.6}$$

where V is the matrix whose columns are formed from the eigenvectors of S_b' in order of the descending values of the corresponding eigenvalues. Finally the transformed, reduced dimension feature vector of Equation 3.2.22 becomes

$$\underline{y}' = W^t \underline{x} \tag{3.3.7}$$

where $\underline{x}$ is the original m+1 dimensional feature vector.

3.4 The Classifier

By using either Equation 3.2.22 or Equation 3.3.7 we can reduce the dimension of the feature vector and also decorrelate their components. Thus the resultant transformed sample space should have fairly well defined boundaries between the classes. For classification, all we need is a simple minimum distance classifier as described in Section 3.1.1. The classifier implemented in this work is:

$$D_i^2 = \sum_{k=1}^{d} (b_k - \bar{y}_{ik})^2$$

where

D_i = total distance for the ith pattern class

b_k = the kth dimension of the transformed unlabelled (input) feature vector

$\bar{y}_{ik}$ = average kth component for the ith class of transformed training vectors.

The classification decision is chosen in favor of the smallest class distance D_i^2. This means that the input vector is categorized as belonging to the pattern class whose average sample it is nearest. Once the classifier has been trained the only information it needs to classify an input object is the transformation matrix and the mean sample transformed feature vector of each expected object boundary.

In this chapter we have shown how the dimension of the feature vector can be reduced by projecting the vector onto a reduced system of K-L coordinate axes. It was also shown how the class mean information can be optimally used to maximize the interset distances. The next chapter will present the necessary hardware and software algorithms to implement the theory of this and the previous chapter.

CHAPTER 4

CREATING THE PATTERN RECOGNITION SYSTEM

This chapter deals with the realization of the operational pattern recognition system and the details of the algorithms used in this system. The first section of the chapter will describe the system hardware and how the original scene is digitized for computer use. The second section discusses the thresholding technique used on the digital representation of the scene. In the third section the algorithms used to extract the raw data from which the features can be selected, are described. The fourth section describes how to extract and use the additional information provided by the multiple edges of an object. The last section describes and provides the flow charts of the software written for this pattern recognition system.

4.1 Object Sensing

We have been assuming that the machine that will do the necessary data manipulation for feature selection and classification is the digital computer. However the computer doesn't have eyes to see the object! We now describe how, by the use of the proper hardware and special software, the computer can receive data just as if it could see.

4.1.1 Description of the System Hardware

The main hardware elements of the object recognition system developed in this work are: 1) the Matsushita Panasonic WV-241 TV camera using the

RS-170 standard, 2) the Image Technology Inc. (ITI) IP-512 series image acquisition board containing the AP-512 Analog Processor and the FB-512 Frame Buffer, 3) the VAX 11-730 digital computer and 4) the Panasonic TR-920M monitor (black and white). See Figure 4.1.1 for the system design.

The TV camera scans the scene and produces an analog, video signal which reproduces the variations in the scene intensity. This analog signal is sent to the ITI image acquisition board where it is flash converted by the circuits of the AP-512 Analog Processor into a data stream of 8 bit bytes. Each byte is a pointer to one of 256 elements of a programmable look-up-table (LUT). If the byte represents the number 8 then the contents of the 8th element of the LUT is put into an appropriate location of a 256-K byte RAM called the FB-512 Frame Buffer. The portion of the frame buffer memory actually used in the system is called the active video window. This window is a 480 row by 512 column two dimensional array of 8 bit numbers (called grey levels) which stores the digitized representation of the complete scene as viewed by the TV camera. Each element of this digitized representation is called a picture element or pixel. Since the TV camera starts its scanning cycle at the top leftmost corner of the scene, the origin of the active video window is also at the top leftmost corner (as seen on the monitor). Figure 4.1.2 shows the origin of the video screen coordinate system of the active video window. The x values increase to the right and the y values increase downward. The contents of the frame buffer are flash converted from digital to analog for real-time viewing on the monitor. By the use of the ITI software applications package, the Fortran programs running on the VAX

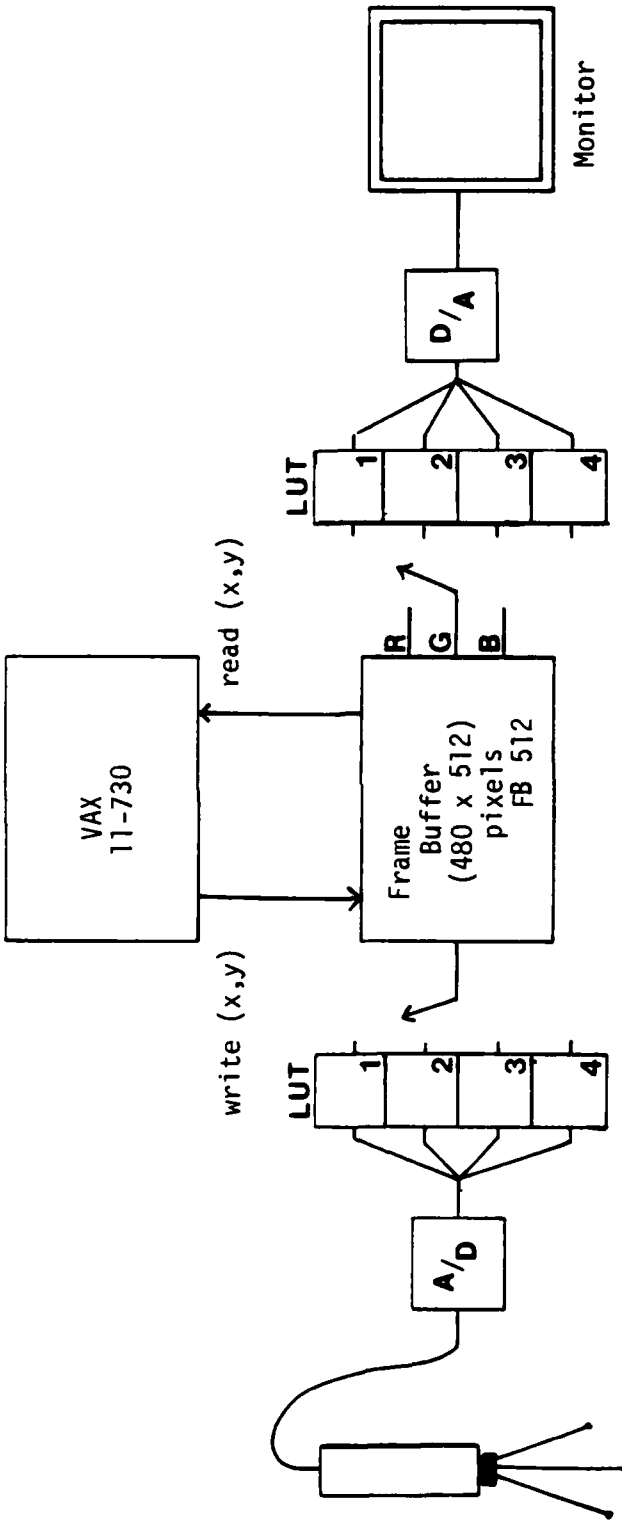


Figure 4.1.1 Block Diagram of the Pattern Recognition System Hardware

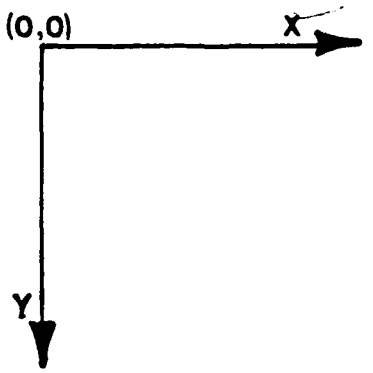


Figure 4.1.2 Active Video Window Coordinate System

computer can access the contents of the frame buffer and the hardware functions of the analog processor.

4.1.2 The Digital Image

In order to provide the computer with data in the form that it can accept, the scene containing the objects is sampled. What we have stored in the frame buffer then is a digital representation of the original continuous scene. Each pixel of this representation is a rectangular element which has an 8 bit value which quantizes the image intensities to any one of 256 gray levels.

As a direct result of this quantized representation formed from these rectangular pixels there arise two problems which must be considered in the implementation of the system. First, the shape of the pixel causes the distance measurements (radius vector lengths), based on pixel location, to vary with changes in object rotation or translation. In the digital representation 4 increments in the x direction have the same physical distance as 5 increments in the y direction. This is referred to as the pixel aspect ratio.

To eliminate the problem due to the rectangular pixel shape, we introduce a pixel aspect correcting factor of 5/4 to the x distance values in our calculations.

A second problem that arises is the effect of the quantization process on the original scene. Instead of the smooth edges which exist in the original scene the representative edges are uneven and sometimes jagged. Also, the number of pixels on the object boundary will vary with slight changes in boundary position. Thus we see variations in the representation of similar objects if they differ slightly in two dimensional translation or rotation. These variations are collectively referred to as quantization noise. The significance of this noise can be reduced by decreasing the size of the pixels or by increasing the relative size of the objects in the representation.

4.2 Preprocessing of the Digital Image

Once we have acquired the digital representation of a scene, we generally like to preprocess the representation so as to make the feature selection algorithms simpler and quicker. The bag of tricks which allows us to perform this preprocessing comes under the heading of Digital Image Processing. However, due to the time constraint on a pattern classifier, the preprocessing of the image is kept to an essential minimum. In this section we will describe the thresholding done to the digital image in order to simplify the boundary detection algorithms.

4.2.1 Image Thresholding

In order to implement a quick and simple boundary detection scheme, the boundaries of the objects must be distinct from the background and they must be uniform in intensity. A simple way to produce these desired properties is to transform the image into a binary picture which contains objects of uniform intensity against a background of a different intensity. The process of producing such a binary image is called thresholding.

To perform this thresholding process we use a single programmable look-up-table on the ITI image acquisition board. On the board there are actually 2 groups of 4 input LUTs, and 3 groups of 4 output LUTs. The 2 input groups allow two possible camera inputs and the 3 output groups allow for the three color groups (RGB of a color system). Each of the four LUTs in any group can be preprogrammed or pre-loaded before actual use. Then during execution, the program can switch LUTs and instantly transform the scene. For our purposes, however, we don't need to transform the scene from moment to moment, but we do need to threshold the image before it is stored in the frame buffer. To do this the 256 locations of the chosen input LUT are loaded with one of two grey-levels. The locations below a certain location (called the threshold) are loaded with one grey level, for example 0, and the locations on and above the threshold are loaded with a different grey level, for instance 255. Imagine that we have a light object on a dark background and a grey scale of 0 (black) to 255 (white). If the LUT has been preloaded as described, then as the scene is initially viewed, the data bytes are immediately converted to either one of the two designated grey levels.

The light areas of the scene where the intensity is above the threshold are stored as grey level 255 in the frame buffer and the dark areas below the threshold are stored as grey level 0.

4.3 Feature Selection

We now assume that we have a binary, thresholded representation of the original scene stored in the frame buffer. We still need, however, to form the time series from the radius vector lengths of a sampled boundary in order to calculate the components of a feature vector.

To do this, an object must first be detected. Once an object is detected, the boundaries need to be traced and the boundary centroids estimated. When we have a centroid pixel position, the radius vector lengths can be calculated and the corresponding time series can be produced. It is only when the complete time series is formed that the AR parameters can be calculated using the algorithm described in chapter two.

In the next pages we will describe the various techniques used to provide the raw measurements from which the features can be selected.

4.3.1 Search for a Boundary Starting Pixel

This pattern recognition system allows the existence of more than one object in the scene viewed by the camera. Also the objects may have more than one closed edge. In order to recognize these multiple edges, it is necessary to scan the entire scene stored in the frame buffer.

To speed up this scanning process not all of the pixels are read. The search for object edges begins at the origin of the active video

window in the frame buffer. The search continues along the constant y row where only the x values which are multiples of a previously prompted for increment are read. Once the end of the row has been reached, the search continues at the beginning of the next y row which is a multiple of the increment. For example, if the incremental step is 8, the following pixels on the first row are read: (0,0), (8,0), (16,0), ... The pixels which are read on the next row are: (0,8), (8,8), (16,8), and so on. The search continues in the same manner until a transition occurs indicating an object has been detected or the bottom right corner of the scene has been reached.

A transition is detected when the difference between the present pixel grey level and the previous grey level is equal to the difference in intensity between the object and the background (in our case 255). Due to the manner of hopping across the rows of the frame buffer in search of objects, there is no guarantee that the pixel where the transition is first encountered is an edge pixel. Thus the search backtracks pixel by pixel on the same row until the transition indicating the actual edge is detected. This pixel where the boundary is first detected is referred to as the start pixel.

4.3.2 The Turtle Boundary Detector

When an object in its two-dimensional representation is of uniform intensity and its edge doesn't contain spurious gaps, then it is possible to use a "turtle" boundary detection algorithm. In this work a modified version of the turtle algorithm used in the work of Dubois [Dubois, 1984] is implemented. In this algorithm the boundary between the object and its background is followed in a clockwise manner by a symbolic device referred

to as the "turtle". Once the boundary start pixel has been found as described in the previous pages the turtle proceeds to trace the edge according to the following two rules [Duda and Hart, 1973]:

1) If the turtle is in the object it will turn left and take a step.

2) If the turtle is in the background it will turn right and then take a step.

The turtle stops when it has returned back to the start pixel.

Since a great portion of the classification time is spent turtling around the boundaries of the objects in the scene it is advantageous to speed up the algorithm. To do this, the turtle algorithm of Dubois [Dubois, 1984] was modified in the following ways:

1) The direction of the turtle movements is governed by decisions based on logical flags indicating past and present turtle position.

2) All subroutine calls have been eliminated in preference for a quicker straight through code.

3) As each boundary pixel is detected, the x and y locations are temporarily stored and counted for later use by the radius vector calculation algorithms.

4) Also, as each boundary pixel is detected, its grey level intensity is changed. This marking of the boundary is to prevent repeat turtling of the edge as the entire frame buffer is scanned. Also the boundary marking provides identification for later determination of which edges are inside other edges. An example of the scene with marked edges is shown in Figure 4.3.1.

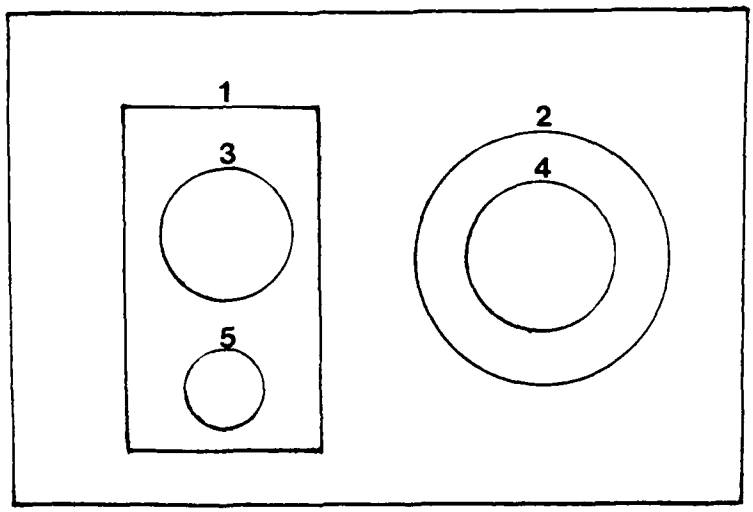


Figure 4.3.1 Example of a scene showing marked edges

Note that the edges are marked with the intensity level which corresponds to their order of detection. For instance the boundary closest to the top left corner is detected first and is thus marked with the grey level intensity 1.

4.3.2.1 The Centroid of the Boundary. The center of the two-dimensional representation of the object can be approximated by calculating the average of the boundary pixel locations. Thus the x_c and y_c coordinates or the boundary center are:

$$x_c = \frac{\sum_{n=1}^{N_b} x_n}{N_b} \quad , \qquad y_c = \frac{\sum_{n=1}^{N_b} y_n}{N_b} \tag{4.3.1}$$

where N_b is the number of boundary pixels and x_n and y_n are the x and y coordinates of the nth boundary pixel. The advantage of finding the boundary centroid is that despite changes in the object's rotation and translation the relative position of the centroid remains the same. The

centroid position can then be used as the origin of a set of reference axes needed for the determination of the boundary intersection radius vector lengths.

4.3.3 Formation of the Time Series from the Radius Vector Lengths

With the position of the boundary centroid known, the two types of time series formed from the radius vector lengths, as described in chapter two, can be produced. The type of time series formed depends on the relationship between each radius vector of the boundary representation. Two methods based on two types of relationships are explored in this work and described below:

4.3.3.1 Equi-angle Method. In this method the intersections of the radius vectors with the boundary are such that the angles between the radius vectors are all constant and equal to $2\pi/N_r$ radians, where N_r is the number of radius vectors. The algorithm actually implemented is a modified version of the algorithm written by Dubois [Dubois, 1984]. The main modification of the original algorithm is the use of the boundary locations stored in an array during initial boundary detection. This eliminates the additional time of turtling around the boundary for a second time. However the trade off is the additional temporary storage requirement for the boundary locations. Another modification is the use of straight through code which also increases the speed of forming the time series.

The basic steps needed to form the desired time series using this method are described next. Initially the origin of a cartesian co-ordinate system is shifted to the centroid of the object boundary, see Figure 4.3.2.

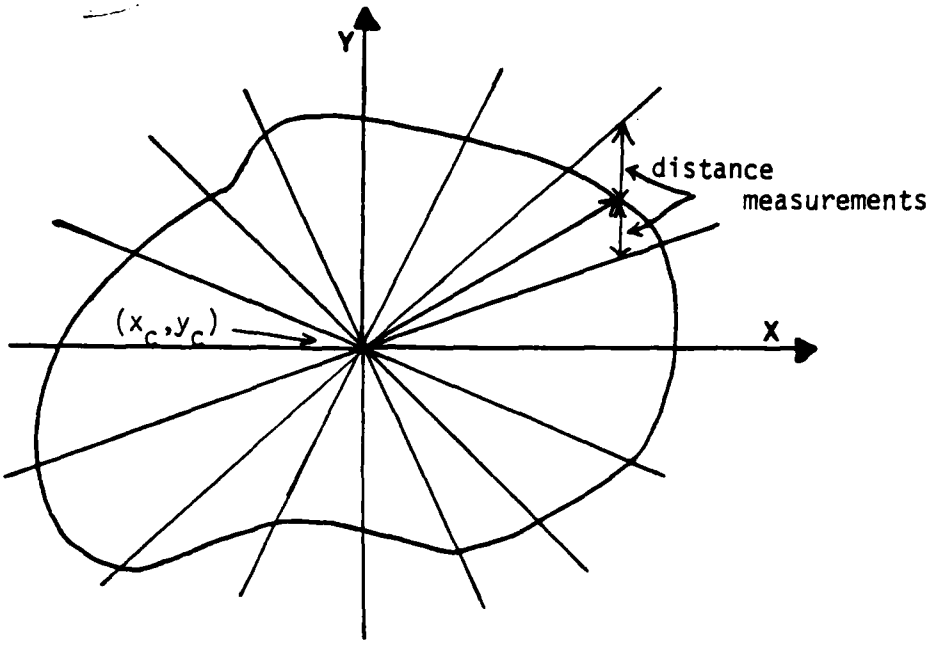


Figure 4.3.2 Example of the Equi-Angle Radius Vector Boundary Intersection Method

The radius vectors are then imagined to project, equally spaced by $2\pi/N_r$ radians, from the origin to the object boundary. Each section formed by a pair of radius vectors is referred to as a sector. Note that in determining the intersection of the radius vector with a boundary pixel, we are only concerned with the relative position of the pixel with respect to the radius vectors of the sector. Thus only the magnitudes of the slopes of the first quadrant of the reference plane need to be calculated. These slopes are calculated once and stored in an array before the actual search for boundary intersections begins. The slope of the jth radius vector is calculated using

$$\text{slope}(j) = 5/4 * \tan(2\pi/N_r * j) \qquad j=0,\ldots,(N_r/4) - 1 \quad (4.3.2)$$

The 5/4 factor is introduced in Equation 4.3.2 to deal with the rectangular pixel shape and to prevent the factor from having to be included in each pixel slope calculation during the search.

The search for radius vector boundary intersections begins with the start pixel. The slope of the imaginary radial line drawn from the start pixel (x_s, y_s) to the centroid pixel (x_c, y_c) is given as:

$$\text{slope} = \frac{|x_s - x_c|}{|y_s - y_c|} \tag{4.3.3}$$

This slope value is compared with the slope values stored in the array and the appropriate sector is found. The two slope vectors of both edges of the sector form reference lines for calculating the relative position of the pixel in the sector. After the start pixel has found its sector and the relative position in the sector has been noted, the second pixel next to the start pixel is checked. Its relative position is noted with respect to the reference vectors also. If the distance measurements, showing the relative pixel position, change signs from one pixel to the next, then it can be assumed that the most recently checked pixel is either on or it has crossed a radial line. The radius vector length between this pixel (x,y) and the centroid can be calculated, i.e.

$$r_t(i) = \sqrt{((x-x_c) * 5/4)^2 + (y-y_c)^2} \tag{4.3.4}$$

This radius vector length becomes part of the time series in the order in which it is detected. This process repeats till all the boundary pixels have been checked and the start pixel has been encountered once again. The details of the implementation of this method are described in the work of Dubois [Dubois, 1984].

4.3.3.2 Equi-Arc Length Method. In this method the intersections of the radius vectors with the boundary are such that the arc length between intersections is constant and equal to a certain number of pixels. In the implementation of this method the constant arc length (A),

measured in pixels, is determined by dividing the total number of boundary pixels (N_b) by the desired number of radius vectors (N_r) i.e.

$$A = \text{arclength} = \frac{N_b}{N_r} \tag{4.3.5}$$

Thus the time series is formed from the radius vector lengths corresponding to every Ath stored boundary pixel where the length from pixel to centroid is calculated using Equation 4.3.4. This method is simpler and quicker than the equi-angle method since it doesn't need to check each boundary pixel for radius vector intersections.

In both radius vector length calculating methods, the (x_{max},y_{max}) position of the boundary intersection pixel of the maximum radius vector is stored. Also the length of the maximum radius vector is stored. From this length we have an indication of the size of shape. Using the position we can calculate the orientation of the maximum radius vector and thereby have an indication of the orientation of entire shape. The orientation of maximum radius vector in the Cartesian coordinate plane is calculated as

$$\text{orientation (degrees)} = \text{Arctan}\left[\frac{-(y_{max}-y_c)}{(x_{max}-x_c)}\right]$$

The negative sign is due to the origin of the video coordinate system being at the top left corner of the scene.

We are now capable of forming two types of time series for any inner or outer object boundary encountered in the scene. Thus it is possible to calculate the AR parameters $\{\theta_{m1},\theta_{m2}, \ldots, \theta_{mm}, \alpha/\sqrt{\beta}\}$ of each boundary at any model order. These parameters can be stored in vector form during the acquisition of training data and transformed later, or they can be transformed immediately for classification purposes. The next section

deals with the extraction of additional information from the inner boundaries of an object and how that information can be used in the classification process.

4.4 Extraction of Additional Information from the Inner Boundaries of an Object

The additional information provided by the inner boundaries of an object can be used to hasten the classification process in many ways. If the number of inner boundaries or holes is unique i.e. no other object has that number of holes, then the classification of that object can be based solely on that number. Also the relative position of the edge used in conjunction with the vector of transformed AR parameters can provide additional information. If the object has more than one hole then it is possible to calculate the AR parameters of the polygonal shape formed by connecting the boundary centroids with straight lines. In this work the AR parameters of the polygon were not calculated but the number of holes and the relative position of the holes inside the object were found. In order to count the number of holes belonging to an object it is first necessary to determine which holes are owned by which object. The technique of determining hole ownership is described next.

4.4.1. Determination of Hole Ownership

Once the boundaries have been detected and marked as in Figure 4.3.1, hole ownership can be determined using the following technique. For each boundary the algorithm is as follows:

1) Begin at the start pixel and proceed to read every pixel on the row to the right of the start pixel.

2) Keep track of any occurrence of a pixel intensity which is neither the object intensity or the background intensity. Once a pixel intensity denoting an edge is detected, ignore any occurrences of the same intensity if they happen to occur immediately after the first encounter.

3) Return to the start pixel and repeat the process on the same row but now going towards the left.

4) Keep a separate record of the occurrences of the edge pixels as in Step 2).

Once both right and left sections of the row containing the start pixel have been read, it is then possible to determine which boundary the edge in question is inside. If the same boundary intensity is detected an odd number of times in both the right and left scans, then it can be assumed that the edge in question is inside the boundary corresponding to the intensity. Figure 4.4.1 shows an example of how this method can be used to determine that edge 2 is inside boundary 1.

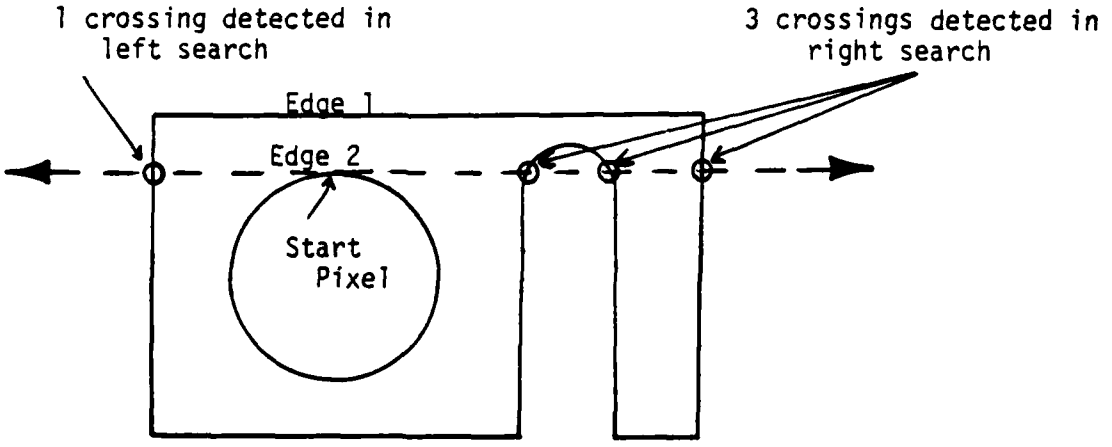


Figure 4.4.1 Example of the Method for Determining Hole Ownership

Once the inner holes of an object have been identified it is a simple matter to count them. It is also possible to order them according to the distance between their centroids and the centroid of the outer boundary. In this work the inner edges were ordered such that the edges whose centroids were furthest away from the outer boundary centroid, were put on the top of the list. When the inner edges have been detected, counted, and ordered, it is then possible to classify an object simply by the hole count or by the classification of each one of its edges.

4.5 Description of System Software

The algorithms described in the preceeding pages were coded into Fortran-77 for the VAX 11-730 computer. The two main programs for the pattern recognition system are called TRAIN and CLASSIFY. Program TRAIN is used to provide the classifier with the training data for each expected object edge. Program CLASSIFY is used to classify objects with one or more edges by the classification of each edge or by the total number of inner edges. An additional program called MAKLIB was written to create a library file (LIB.DAT) of AR parameters of the edges. This library file increased the speed of system tests since it eliminated the need for recalculating the AR parameters of the edges for each test. A complete listing of TRAIN and CLASSIFY and the subroutines they call can be found in the Appendix at the end of this work.

4.5.1 Program TRAIN

Program TRAIN is an interactive program which trains the classifier with the data of the edges of the expected objects. This program produces a TRAIN.DAT file which contains the M+1 columns of the trans-

formation matrix and also the average of the transformed samples of each class. Multiple edges are allowed in the scenes, however all edges of the objects in view must be identical. [See Figure 4.5.1 for the general flowchart of program TRAIN.] The program begins by initializing the ITI hardware and then prompts the user for the scan step, the AR model order, and the number of radius vectors. It also prompts for the desired transformation technique which can be either the basic Karhunen-Loeve (KLSEL), the minimum entropy K-L (MESEL), or the optimal prewhitened K-L (PREKL) coordinate axes generating technique. Once the scene has been viewed, thresholded, and frozen, it is scanned in order to find the start pixel of each boundary. As soon as a boundary start pixel is detected, the entire boundary is traced and the centroid is calculated. After the entire scene has been scanned, the radius vector lengths of each boundary are determined so that the AR parameters can be calculated. Once the feature vectors are formed from the AR parameters, the process repeats and another scene can be viewed. When enough samples of each class have been accumulated, all the AR parameter feature vectors are passed to the desired subroutine which then forms the average covariance matrix and finds the particular K-L coordinate axes. After the K-L axes are calculated and the transformation matrix is formed, the original feature vectors are transformed and the average transformed vectors are found. These vectors are stored along with the transformation matrix in the training data file.

4.5.2 Program CLASSIFY

Program CLASSIFY is an interactive program which can recognize objects with one or more edges. The program allows many different objects to be present in the scene. It also calculates the rotation of the objects. The

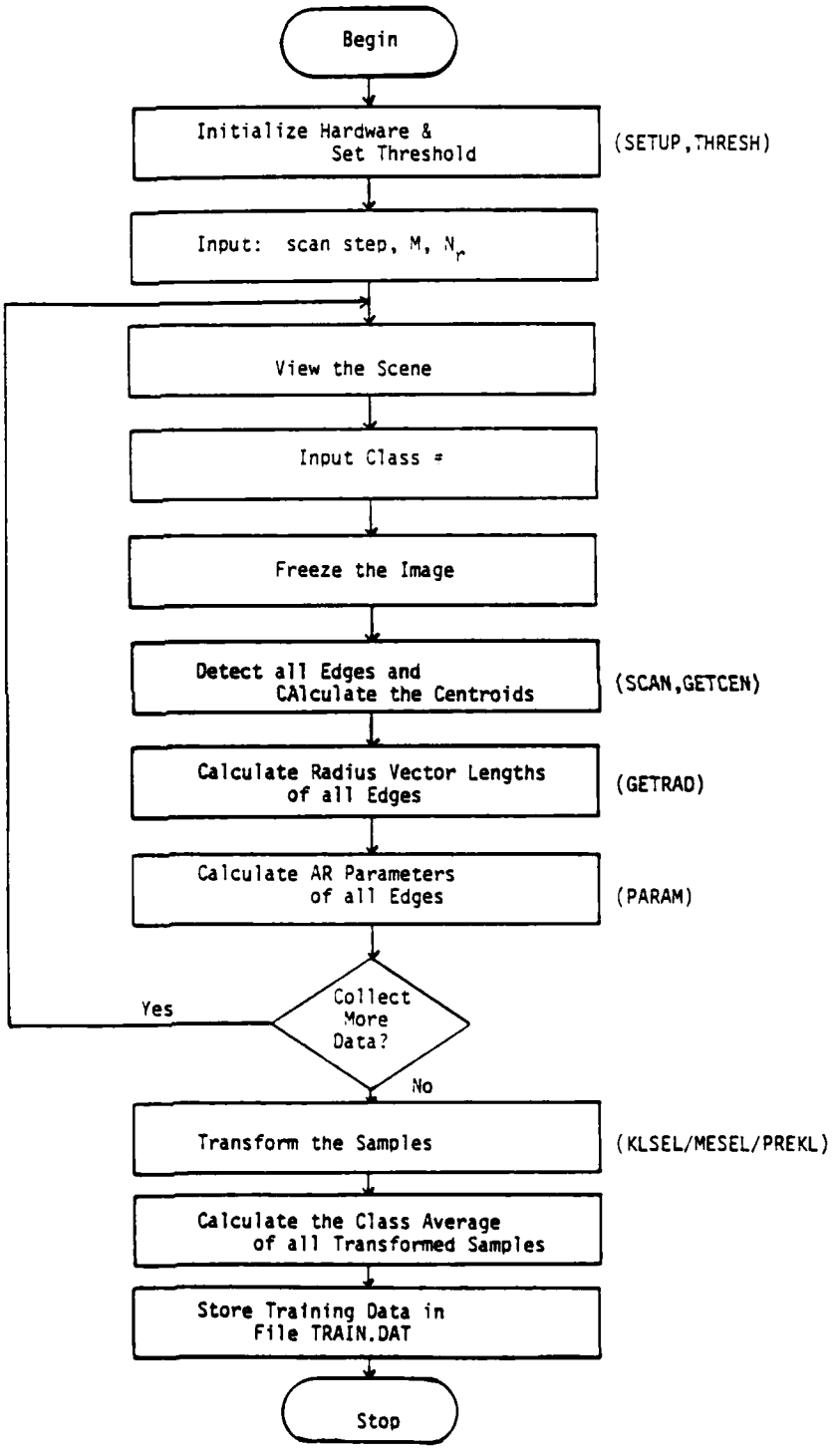


Figure 4.5.1 General Flowchart of Program TRAIN

program starts by initializing the hardware and prompts for the desired constants as in Program TRAIN. It also prompts for the desired dimension of the average transformed feature vectors. Once the constants have been accepted, the program scans the frame buffer and turtles around all edges, calculating the centroid after detecting each edge. When all the edges have been detected and also marked, the program then determines which holes are owned by which objects. The inner holes of the objects are counted and if the count is unique, then the object can be classified. The edges of the yet unclassified objects now need to be classified so the radius vector lengths are determined and the AR parameters are calculated. Then using the transformation matrix provided by the training program, the AR parameters of each edge are transformed. Now the transformed feature vectors can be classified using the minimum-distance algorithm as discussed in Chapter 3. Once all the edges of the object are classified, the object can be classified. This is done by matching the edge classifications and the number of holes data with the object data contained in a file called OBJECT.DAT. This data file is created by the user in the same way that an ordinary text file is created. The object data is typed in one line per object. For example, suppose we have an object with label 3 which has 2 holes labelled 5 and 7. If the centroid of hole 7 is further away from the outer boundary centroid than hole 5, then we would type in the sequence: 3 2 7 5. Once all the objects have been classified, the program is ready to accept another scene full of objects or the session can be terminated. See Figure 4.5.2 for the general flowchart of Program CLASSIFY.

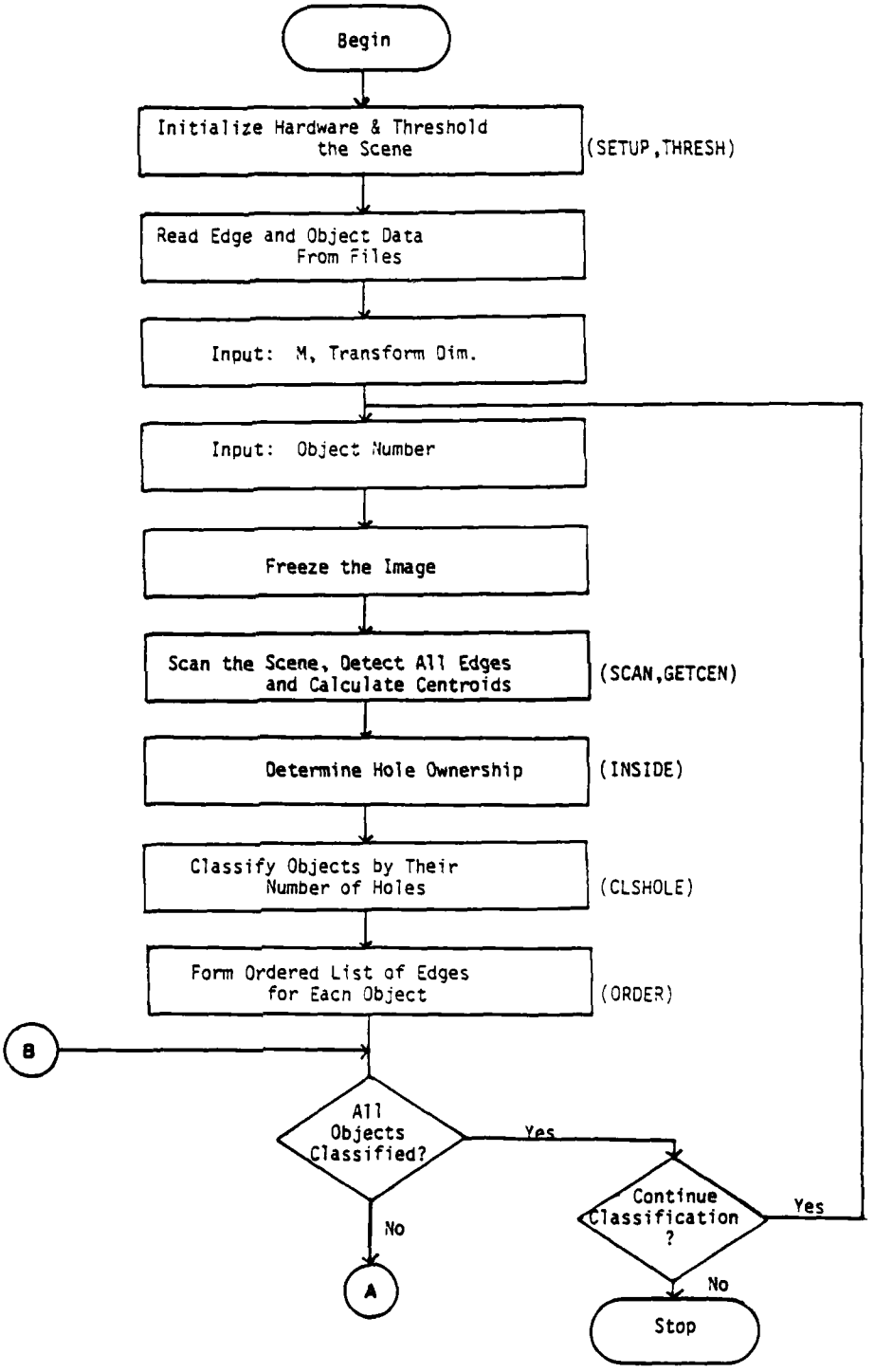


Figure 4.5.2 General Flowchart of Program CLASSIFY

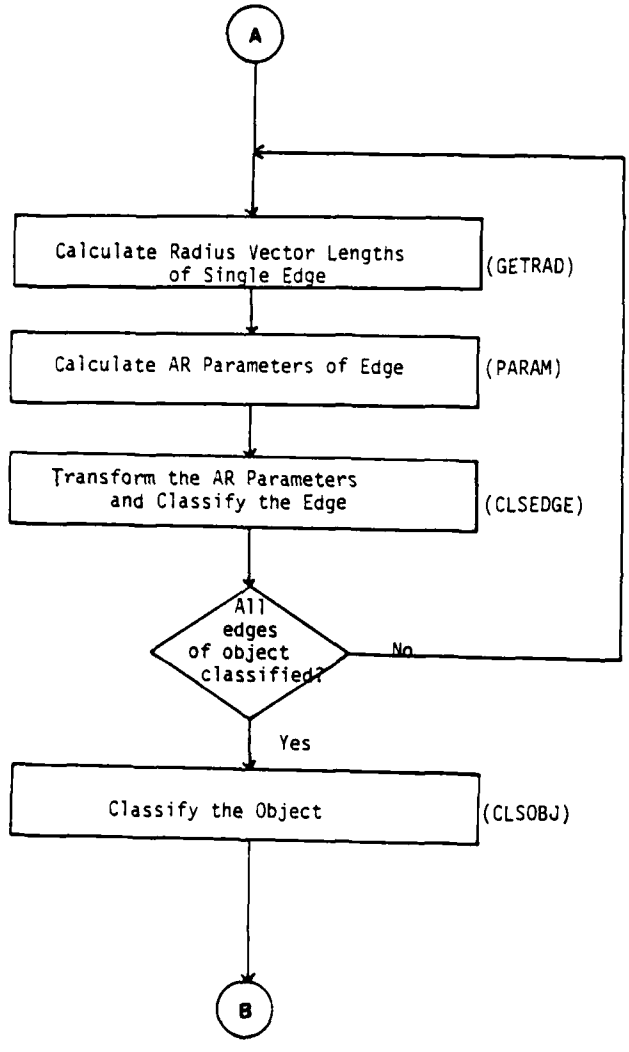


Figure 4.5.2 Continued

CHAPTER 5

NUMERICAL EXAMPLE AND SYSTEM TESTS AND RESULTS

In the previous chapters, we described the theory and implementation of our pattern recognition algorithms. It is well known that any theoretical study is incomplete without examples and experimental tests. Thus this chapter is divided into two parts, the first part provides a numerical example of the feature selection, transformation, and classification theory. The second part describes the series of tests and their results, of the entire recognition system. In all examples and system tests the image threshold was set at 120, the scan step was 8 pixels, and the desired number of radius vectors was 64.

5.1 Numerical Example

We used programs TRAIN and CLASSIFY to provide for the following examples of the theories and algorithms used in this pattern recognition system. Two examples have been provided, the first is of the system using the equal angle boundary sampling method. The second example shows the system using the equal arc length boundary sampling method. The shapes chosen for these examples are the familiar S (class 1) and N (class 2) used in the work of Dubois [Dubois, 1984]. These shapes were chosen for two reasons. The first is that the AR parameters of both classes are very similar and thus they show the need for a type of transformation technique. Also they were chosen to allow quick comparison of the two different pattern recognition systems. The actual size of these shapes

are considerably larger than normal text. The letter S is approximately 6 inches tall and the N is about 4-1/2 inches tall.

5.1.1 Example of the System Using the Equal Angle Boundary Sampling Method

The equal angle boundary sampling technique described in Chapter Two is used in this example. Figure 5.1.1 shows the sampled shapes of class 1 and class 2. Note that portions of the letters are not sampled as would be expected. This is due to the radius vector boundary intersection algorithm rejecting radius vector measurements if they occur on the same slope as the previous radius vector length measurement. This rejection was done to prevent the occurrence of a disproportionately large number of similar radius vectors measurements if the slope of the boundary happened to coincide with the slope of a radius vector. Below the shape of each letter is the corresponding time series formed from the radius vector lengths. Notice the fast vertical changes. These unnatural transitions are the direct result of the sampling process skipping or by-passing certain sections of the boundary. These quick transitions increase the high frequency content of the time series wave shape and thus it can be expected that the resultant AR model order will be higher for this type of wave shape than for a waveform that doesn't have these sharp transitions.

Table 5.1.1 shows the original training set data comprising 10 samples of each class. As can be seen, the samples varied in size and rotation. The size variation can be observed in the changes of the length (in pixels) of the maximum radius vector from one sample to the next. Notice also that the changes in rotation are not strict multiples of $2\pi/N_r$. This was done to mimic a realistic setting where the position and rotation of an object is generally completely arbitrary.

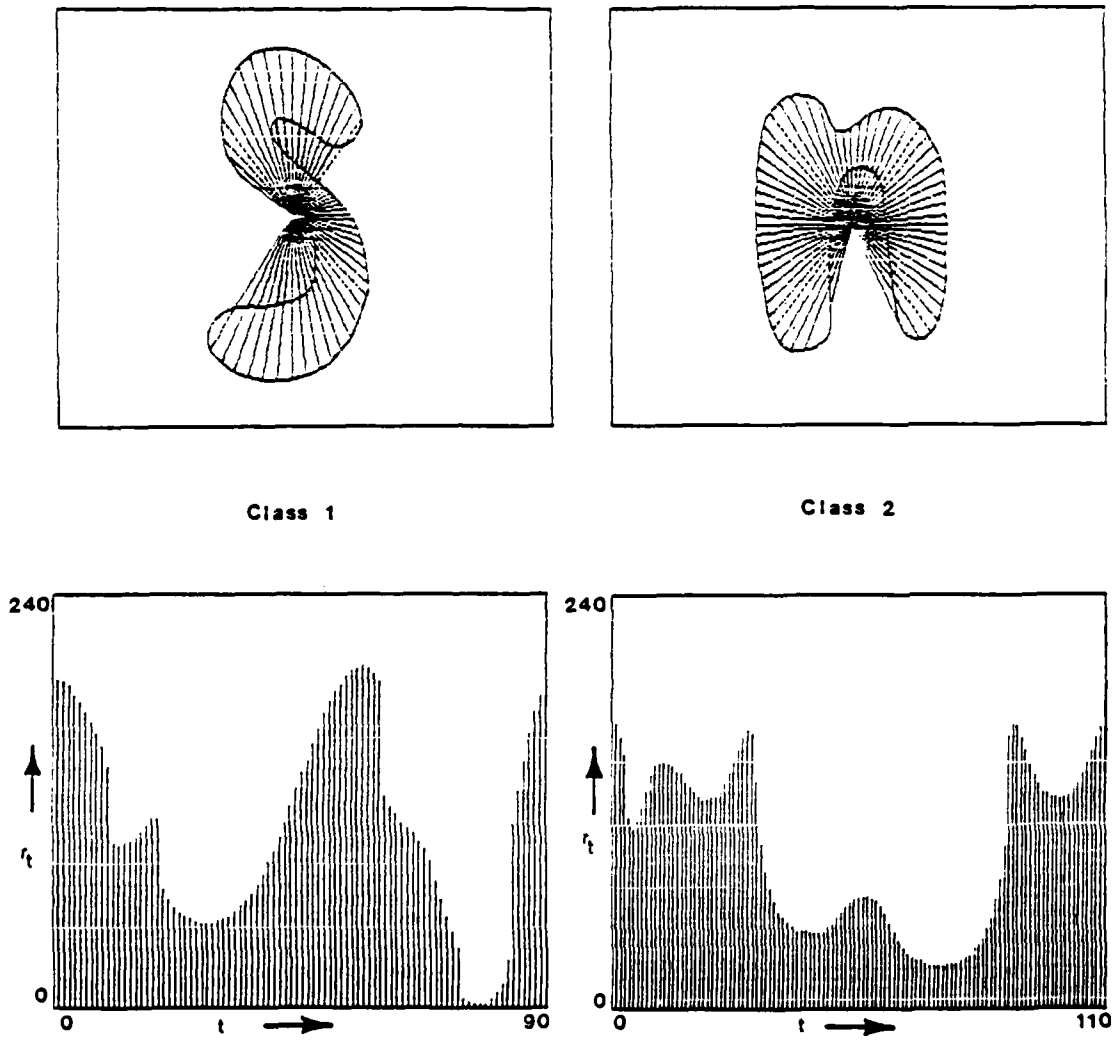


Figure 5.1.1 Shapes for Numerical Example: Equal Angle Boundary Sampling Method

Table 5.1.1 Training Set Data

Equal Angle Sampling Method

(AR Model Order: 1)

Sample	N_r	θ_1	$\alpha/\sqrt{\beta}$	Length of Max Rad Vec.	Rotation
			Class 1		
1	95	-0.97653	1.73521	183	251.4
2	88	-0.97261	1.8513	187	207.1
3	84	-0.96371	1.82773	188	166.0
4	88	-0.97306	1.80995	187	339.0
5	90	-0.96508	1.79821	193	313.8
6	90	-0.96442	1.86203	196	56.5
7	80	-0.95070	2.10665	203	13.7
8	84	-0.96093	1.81376	204	339.0
9	91	-0.97108	1.67467	214	326.0
10	86	-0.91653	1.92008	209	278.0
			Class 2		
1	110	-0.97338	1.91497	160	236.0
2	110	-0.97753	1.96800	165	193.8
3	113	-0.97996	1.88319	168	159.2
4	112	-0.97825	1.91980	166	326.0
5	112	-0.98253	1.96457	174	303.1
6	110	-0.97322	1.93067	177	39.7
7	113	-0.97945	1.94201	182	345.9
8	113	-0.97759	1.92446	190	20.6
9	110	-0.97306	1.93831	186	128.5
10	111	-0.97896	1.95099	188	213.5

The AR parameters θ_1 and $\alpha/\sqrt{\beta}$ cluster around their average class value as can be seen in Figure 5.1.2a. However looking at Table 5.1.1, some θ_1 values and also some $\alpha/\sqrt{\beta}$ values overlap between the classes. Thus it can be seen that it is desirable to reduce or possibly eliminate these overlaps by using a transformation technique. Table 5.1.2 shows the results of transforming the original AR parameters using the basic K-L, the minimum entropy K-L and the optimal K-L transformation matrices as described in Chapter 3. The plots of these transformed points are shown in parts b,c,d of Figure 5.1.2.

To understand how the K-L coordinate axes can reduce the dimension of the feature vector, observe how the transformation rotates the original coordinate axes so they now point in the direction of maximum and minimum variance. In the basic K-L transformed system, the first axis or x axis, points in the direction of maximum within class variance. Looking at plot (b) in Figure 5.1.2, it is noticed that the major axis of the ellipsoid formed from the cluster of class 1 points of plot (a) is now parallel with the x axis. We can also see that along the y axis corresponding to the direction of minimum variance, there is no overlapping between the classes. It is possible to draw a straight line parallel to the x axis between the class clusters such that the classes do not overlap. An unlabelled input feature vector can be classified by its position along the y axis alone. Thus the dimension of the feature space has been reduced.

Plot (c) in Figure 5.1.2 shows the effect of the minimum entropy transformation. Comparing this plot with plot (b) we see that plot (c) is the same as (b) but with the axes reversed. In the minimum entropy transformed system, the x axis points in the direction of the minimum

Table 5.1.2 Transformed Training Data

Sample	Basic K-L		Min Entropy K-L		Optimal K-L	
	T1	T2	T1	T2	T1	T2
			Class 1			
1	1.685	1.061	1.061	1.685	263.35	54.11
2	1.801	1.063	1.063	1.801	269.21	52.84
3	1.778	1.053	1.053	1.778	266.64	52.41
4	1.760	1.062	1.062	1.760	268.67	53.23
5	1.748	1.053	1.053	1.748	266.51	52.76
6	1.812	1.056	1.056	1.812	267.35	52.17
7	2.057	1.054	1.054	2.057	267.81	49.09
8	1.764	1.050	1.050	1.764	265.73	52.33
9	1.625	1.053	1.053	1.625	266.06	54.25
10	1.870	1.056	1.056	1.870	267.55	51.46
			Class 2			
1	1.864	1.067	1.0671	1.864	270.40	52.35
2	1.917	1.074	1.074	1.917	272.25	52.18
3	1.832	1.072	1.072	1.832	271.52	53.08
4	1.869	1.072	1.072	1.869	271.68	52.65
5	1.913	1.079	1.079	1.913	273.43	52.56
6	1.880	1.067	1.067	1.880	270.60	52.20
7	1.891	1.075	1.075	1.891	272.32	52.54
8	1.874	1.072	1.072	1.874	271.58	52.56
9	1.888	1.068	1.068	1.888	270.68	52.12
10	1.900	1.075	1.075	1.900	272.34	52.43

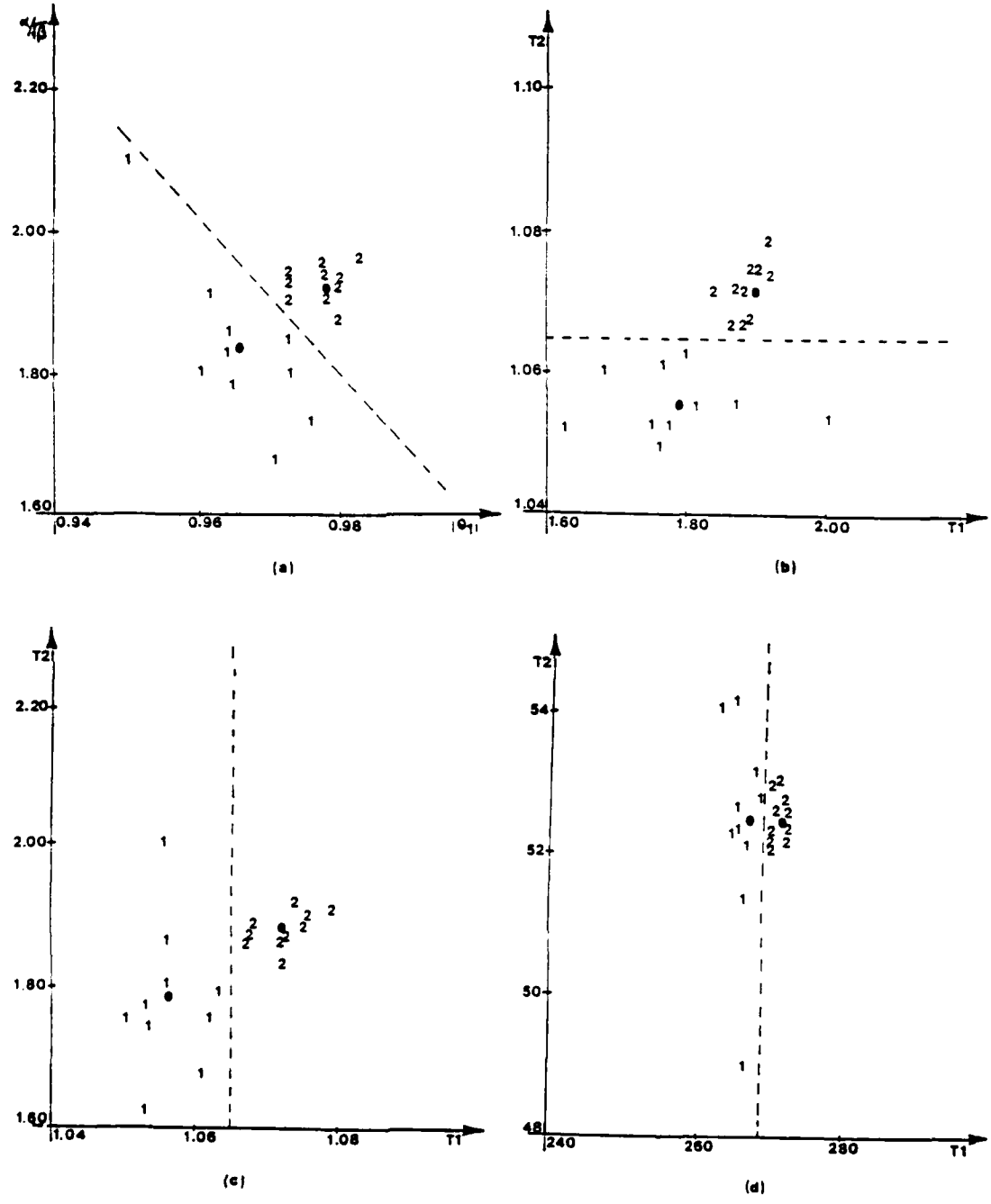


Figure 5.1.2 Plots Showing the Effects of the Different Transformation Techniques: (a) Original AR Parameters, (b) Basic K-L Transformed Parameters, (c) Minimum Entropy K-L Transformed Parameters, (d) Optimal K-L Transformed Parameters

within class variance. This is to be expected since the transformation matrix is formed from the exact same eigenvectors as the K-L transformation matrix, but reversed in order. Since this reversal of eigenvectors does not enhance the basic K-L system, the minimum entropy transformation was not investigated further. However, if it is known a priori that all the class covariance matrices of a training set are equal, then it is shown [Tou and Gonzalez, 1974] that the minimum entropy method produces better results than the basic K-L transformation.

Plot (d) in Figure 5.1.2. shows the effect of the optimal K-L transformation. The x-axis points in the direction of maximum between class variance. Now the class means fall on a line parallel to the x-axis. It is also noted that the intraset distances along the x-axis have been reduced in comparison to the original clusters. Thus any unlabelled input vector can be categorized according to its position along the x axis.

In this 2 class problem, we do not observe a distinct advantage of the optimal K-L transformation. However, as the feature space increases and includes more classes the ability of the basic K-L transformation to divide the feature space decreases. This is because the K-L transformation is based on the within class variances which do not contain a great amount of discriminatory information. On the other hand the optimal K-L transformation uses the prewhitened between class variances and thus maximizes the use of the discriminatory information contained in the class means.

Tables 5.1.3 and 5.1.4 show the data of the classification process. The classifier initially retrieves from storage the transformation matrix, which was used to transform the training samples, and the

Table 5.1.3 Classification Data Using Basic K-L Transformation

(AR Model Order: 1, Transformed Dimension: 2)

INPUT TRAINING DATA

Transformation Matrix:

0.04958	-0.99877
0.99877	0.04958

Mean Transformed Training Samples:

	T1	T2
Class 1:	1.78981	1.05600
Class 2:	1.88286	1.07206

CLASSIFICATION DATA

θ_1	$\alpha/\sqrt{\beta}$	T1	T2	Length of Max Rad Vec	Rotation	Decision
			Class 1			
-0.95892	1.85477	1.805	1.050	190	66.8	1
-0.96278	1.76381	1.714	1.050	195	27.0	1
-0.96952	1.69998	1.650	1.053	195	0.0	1
-0.97158	1.88558	1.835	1.064	192	326.2	1
-0.97213	1.75851	1.708	1.058	209	319.2	1
-0.97280	1.84680	1.796	1.063	204	122.9	1
-0.97630	1.67335	1.623	1.058	198	246.8	1
-0.97096	1.80878	1.758	1.059	201	213.3	1
-0.97648	1.68312	1.632	1.059	221	213.7	1
-0.96265	1.98290	1.933	1.074	219	146.1	2
			Class 2			
-0.97963	1.92738	1.876	1.074	180	219.8	2
-0.98326	1.89282	1.842	1.076	186	172.7	2
-0.97366	1.91872	1.868	1.068	185	7.0	2
-0.97945	1.92982	1.878	1.074	183	314.0	2
-0.98020	1.94440	1.894	1.075	198	313.8	2
-0.97278	1.94115	1.891	1.067	197	39.8	2
-0.98261	1.95360	1.902	1.078	200	338.9	2
-0.97505	1.93480	1.884	1.070	206	13.9	2
-0.97879	1.91234	1.861	1.072	208	339.0	2
-0.97175	1.91964	1.869	1.066	205	123.2	2

CONFUSION MATRIX

	1	2	
1	9	1	95.0% correct
2	0	10	

Table 5.1.4 Classification Data Using Optimal K-L Transformation

(AR Model Order: 1, Transformed Dimension: 2)

INPUT TRAINING DATA

Transformation Matrix:

$$\begin{bmatrix} -246.90 & -70.75 \\ 15.70 & -8.63 \end{bmatrix}$$

Mean Transformed Training Samples:

	T1	T2
Class 1:	267.388	52.467
Class 2:	271.682	52.467

CLASSIFICATION DATA

θ_1	$\alpha/\sqrt{\beta}$	T1	T2	Length of Max Rad Vec	Rotation	Decision
			Class 1			
-0.97576	1.79706	269.13	53.53	183	260.8	1
-0.97308	1.81335	268.73	53.20	184	231.3	1
-0.95263	2.00491	266.69	50.10	186	186.9	1
-0.95786	1.82341	265.13	52.04	186	139.8	1
-0.97406	1.79275	268.65	53.45	181	94.4	1
-0.96344	1.84091	266.78	52.28	196	71.7	1
-0.96077	1.87224	266.61	51.82	200	39.8	1
-0.95672	1.90406	266.11	51.26	201	345.8	1
-0.97174	1.71986	266.93	53.91	204	326.0	1
-0.95385	2.16893	269.56	48.77	200	288.2	2
			Class 2			
-0.97945	1.93141	272.15	52.63	170	134.2	2
-0.97635	1.97708	272.10	52.02	172	200.7	2
-0.97666	1.88894	270.80	52.80	173	186.6	2
-0.97523	1.87643	270.25	52.81	172	152.5	2
-0.98230	1.93227	272.87	52.83	174	0.0	2
-0.98010	1.93268	272.33	52.67	170	297.8	2
-0.97795	1.98704	272.66	52.05	187	33.7	2
-0.97538	1.93191	271.16	52.34	189	346.0	2
-0.98182	1.93898	272.86	52.73	200	288.2	2
-0.97969	1.94883	272.49	52.50	198	123.2	2

CONFUSION MATRIX

	1	2	
1	9	1	95.0% correct
2	0	10	

formed training sample of each class. The table shows the original AR parameters and the transformed parameters of the test set. The maximum radius vector length and the rotation of the object are also listed. Like the training samples, the test samples are random. The resultant classifier decisions are shown in the right column. For both sets of classification data, the basic K-L transformed and the optimal K-L transformed, the classifier recognized the input shape correctly for all test samples except one. This misclassification occurred in both sets for a sample in which the original AR parameters vary more than the rest. It is suspected that this difference is due to a large segment of the boundary not being sampled. It is expected that the classification performance at this low model order (i.e. M=1) will be improved by the equal arc length boundary sampling method. This will be investigated next.

5.1.2 Example of the System Using the Equal Arc Length Sampling Method

The equal arc length boundary sampling method as described in Chapter Two is used in this example. Figure 5.1.3 shows the sampled shapes of class 1 and class 2. Below each sampled shape is the resultant time series. As can be observed in these time series plots, the sharp vertical transitions, which occurred in the time series of the previously discussed sampling method, do not occur. The waveforms show the gradual changes in the contour of the boundary as it is sampled. Tables 5.1.5 and 5.1.6 show the training and classification data. For this example, only the optimal K-L transformation was used. As expected, the AR parameters and thus the transformed parameters cluster closely about their respective class means. As a result we see a perfect

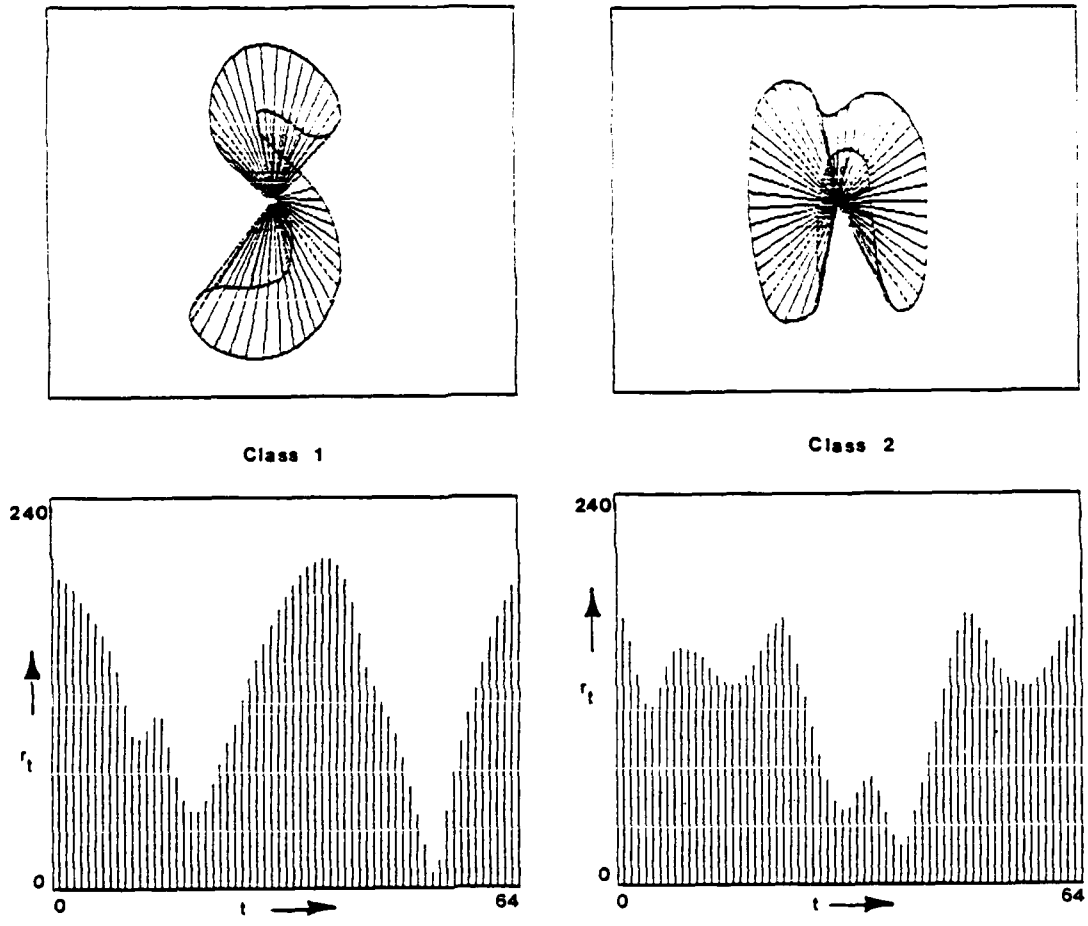


Figure 5.1.3 Shapes for Numerical Example: Equal Arc Length Boundary Sampling Method

Table 5.1.5 Training Set DATA Using the Equal Arc Length Sampling Method and the Optimal K-L Transformation

(AR Model: 1)

Sample	θ_1	$\alpha/\sqrt{\beta}$	Length of Max Rad Vec	Rotation	Transformed Samples (Optimal K-L) T1	T2
			Class 1			
1	-0.97331	2.26854	202	229.5	-270.4	1479.1
2	-0.97314	2.31667	203	193.8	-271.4	1479.9
3	-0.97040	2.34302	200	144.7	-271.4	1476.6
4	-0.97041	2.46447	204	329.6	-274.3	1479.4
5	-0.97226	2.38723	219	78.9	-272.9	1480.3
6	-0.97322	2.29049	217	30.7	-270.8	1479.5
7	-0.97209	2.35664	216	348.4	-272.1	1479.4
8	-0.96968	2.44291	220	299.1	-273.6	1477.8
9	-0.97061	2.43155	225	280.3	-273.5	1478.9
10	-0.97228	2.30699	183	26.0	-271.0	1478.5
			Class 2			
1	-0.96430	2.89333	184	111.5	-282.9	1480.3
2	-0.96269	2.94088	188	180.0	-283.6	1478.9
3	-0.96187	2.91875	188	137.5	-282.9	1477.3
4	-0.96357	2.88612	187	335.1	-282.6	1479.0
5	-0.96345	2.91985	186	78.7	-283.3	1479.6
6	-0.96316	2.90567	199	270.6	-282.9	1478.9
7	-0.96178	2.95423	203	336.4	-283.7	1477.9
8	-0.96132	2.99844	199	296.0	-284.7	1478.3
9	-0.96314	2.89153	204	276.2	-282.6	1478.5
10	-0.96405	2.92276	202	113.8	-283.5	1480.6

Table 5.1.6 Classification Data Using the Equal Arc Length Sampling Method and the Optimal K-L Transformation

(AR Model Order 1: Transformed Dimension: 1)

INPUT TRAINING DATA

Transformation Matrix:

223.52	-1466.18
-23.27	22.96

Mean Transformed Training Samples:

	T1	T2
Class 1:	-272.15	1478.9
Class 2:	-283.27	1478.9

CLASSIFICATION DATA

ϑ_1	$\alpha/\sqrt{\beta}$	T1	Length of Max Rad Vec	Rotation	Decision
		Class 1			
-0.97266	2.35135	-272.14	186	260.9	1
-0.97328	2.28844	-270.81	188	231.0	1
-0.97302	2.29317	-270.86	190	192.4	1
-0.97031	2.40888	-272.95	186	146.9	1
-0.97180	2.33380	-271.54	197	147.7	1
-0.97035	2.44807	-273.87	196	317.8	1
-0.97194	2.36741	-272.35	198	77.9	1
-0.97324	2.29135	-270.87	200	26.4	1
-0.97202	2.28805	-270.52	208	27.5	1
-0.97262	2.30752	-271.11	206	346.5	1
		Class 2			
-0.96444	2.88873	-282.81	170	289.9	2
-0.96061	2.93651	-283.06	177	8.9	2
-0.96100	2.93754	-283.17	176	321.7	2
-0.96298	2.89563	-282.64	170	276.5	2
-0.96398	2.91838	-283.39	169	242.3	2
-0.96461	2.88745	-282.82	179	114.0	2
-0.96244	2.91625	-283.00	183	199.2	2
-0.96240	2.94407	-283.64	184	142.2	2
-0.96264	2.91241	-282.96	191	12.8	2
-0.96283	2.92158	-283.21	190	295.6	2

CONFUSION MATRIX

	1	2
1	10	0
2	0	10

100% correct

classification performance using only one dimension of the transformed space. This example shows how the equal arc length sampling method produces excellent results at a low AR model order.

5.2 System Tests and Results

We used programs TRAIN, CLASSIFY and MAKLIB to test the performance of the pattern recognition system. We tested the system with different shape categories and with varying numbers of shapes in each category. In all the tests the training data for all the desired model orders was acquired from a library data file containing the AR parameters for model orders 1-10 for the samples of the particular shape category. Thus for each shape category the training data for model orders 1 through 10 was formed from the same set of samples.

For each test we have provided a table of results showing the overall percent correct classifications of the shapes category at a certain transformed dimension and particular model order. The percent calculation value was calculated as the average of the class percent correct classifications. This method of calculating percentages downgrades the classifier more if the misclassifications occur for one particular class than if the misclassifications occurred randomly among all the classes.

All the shapes used in the tests were cut out of white paper and viewed against a black background. This was done to eliminate the effects of shadows, reflections, and texture on the performance of this pattern recognition system. In all the tests the objects were translated and rotated randomly in the scene. Also the camera was lowered and elevated randomly between samples to change size of the viewed object. In all the tests, except the multi-edge object and timed tests, the

classification data was provided by the same library file as was used in the training process. This was done to speed up the tests since the AR parameters could simply be pulled out of storage instead of calculated. We felt that the system would still give accurate results since the only data the classifier uses in its decision making process is the mean class transformed training samples where the means are calculated over 25 samples.

5.2.1 Industrial Shapes Test

This was the first test done and it was used to compare the basic K-L vs the optimal K-L transformation techniques. We chose this set to again allow comparison of our system with the system of Dubois. The 8 industrial shapes (see Figure 5.2.1) in this test are scaled down versions of the industrial shapes tested by Dubois. These shapes were originally collected from the literature concerning industrial shape pattern recognition.

Table 5.2.1 shows the classification results of the system using both the basic K-L transformation and the optimal K-L transformation. As can be observed the basic K-L transformed system produces classification rates of 91.5 to 98.5 percent at the maximum transformed dimension for model orders 1 through 10. A significant improvement is seen in the classification results of the optimal K-L transformed system. At model orders 3 through 10 we see perfect classification rates and for model orders 5,7, and 9 these perfect rates occur when using only two dimensions of the transformed feature vector space. Generally the misclassifications occurred when the classifier confused objects 1 and 8 and objects 4 and 5.

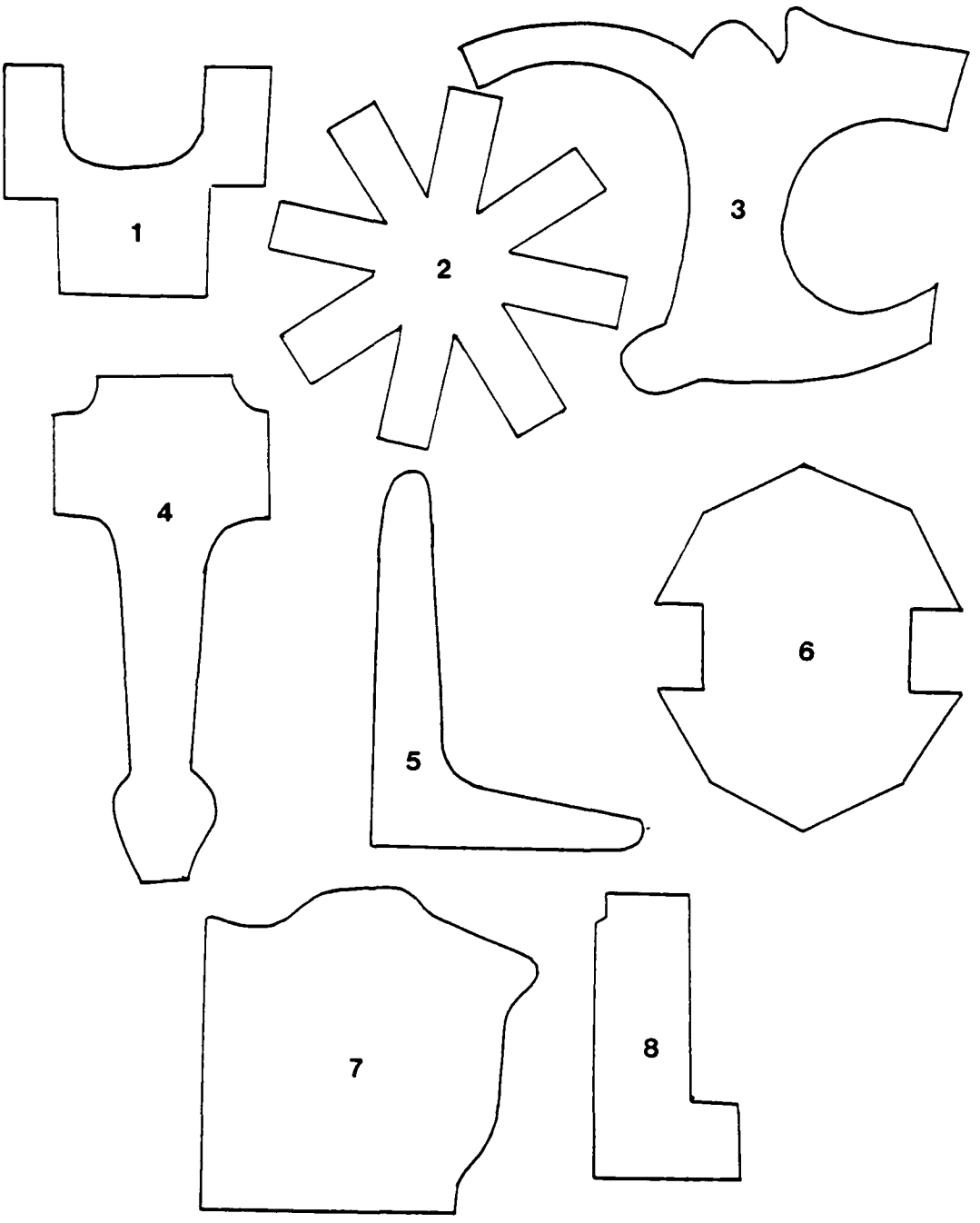


Figure 5.2.1 Industrial Shapes

(reduced to 74%)

Table 5.2.1 Results of the Industrial Shapes Test Using the Basic K-L and the Optimal K-L Transformation Matrices

(25 Test Samples = 25 Training Samples)

% Correct Classification

Model Order	Transformed Dimension 1	2	3	4	5	6	7	8	9	10
				BASIC K-L						
1	96.0	96.0								
2	91.5	92.0	97.0							
3	90.5	91.0	93.5	93.5						
4	94.5	91.0	91.0	91.5	91.5					
5	89.0	90.0	95.5	96.0	96.0	96.0				
6	49.0	87.5	88.5	91.5	91.5	91.5	91.5			
7	62.5	89.5	89.5	96.5	96.5	98.0	98.5	98.5		
8	58.5	78.0	80.5	89.0	93.5	97.5	97.5	97.5	98.0	
9	83.0	84.5	86.0	84.0	92.0	94.0	96.0	96.5	97.0	97.0
				OPTIMAL K-L						
1	85.0	94.5								
2	87.5	98.0	99.0							
3	86.5	95.5	100.0	100.0						
4	80.0	94.5	100.0	100.0	100.0					
5	77.0	100.0	100.0	100.0	100.0	100.0				
6	84.5	99.0	100.0	100.0	100.0	100.0	100.0			
7	96.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0		
8	88.0	99.0	98.5	99.0	100.0	100.0	100.0	100.0	100.0	
9	91.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
10	90.5	98.0	99.5	100.0	100.0	100.0	100.0	100.0	100.0	100.0

5.2.2 Military Shapes Test

This test checked the system performance on a category of 4 military shapes as shown in Figure 5.2.2. One aspect of this test was to demonstrate how this system could be used for military target classification. Also, since all the silhouettes are long and pointed, another aspect was to see how the system would perform in the classification of fairly similar shapes. The optimal K-L transformation matrix was used and both boundary sampling methods were implemented in this test.

The test results in Table 5.2.2 show that the classifier performed perfectly using the equal angle sampling method and 1 dimension of the transformed feature space. The classifier performed slightly less well for the equal arc length sampling method. The misclassifications occurred when the classifier confused shapes 1 and 3. Note that shape 1 is the smallest of the shapes in this category.

5.2.3 Geometric Shapes Test

Many two dimensional representations of objects are fairly geometrical in shape, i.e. the shapes are symmetric and composed of lines and curves. Thus a test was performed on a set of 8 geometric shapes (see Figure 5.2.3). This test also provided the individual edge classification results which could then be compared with the results of the next test where the edges are combined to form multiple edge shapes. As in the military shapes test, the optimal K-L transformation matrix and both boundary sampling methods were implemented.

Table 5.2.3 shows the classification results. Note that the classification rate ranges from 93.5% to 99.5% for the maximum transformed

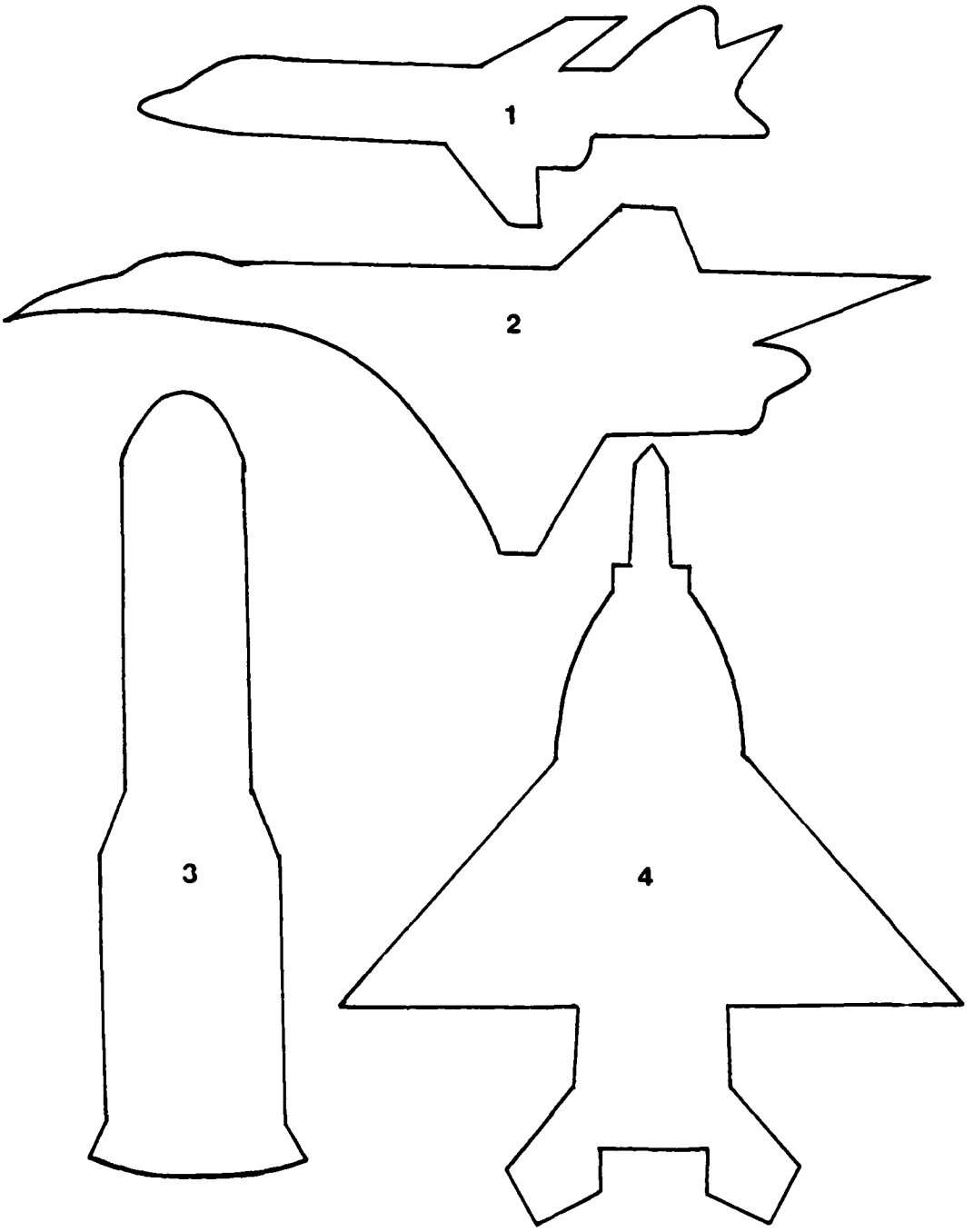


Figure 5.2.2 Military Shapes

(original size)

Table 5.2.2 Results of the Military Shapes Test Using the Optimal K-L Transformation and Both Boundary Sampling Methods (25 Test Samples = 25 Training Samples)

% Correct Classification

Model Order	Transformed Dimension 1	2
EQUAL ANGLE SAMPLING METHOD		
1	100.0	100.0
2	100.0	100.0
EQUAL ARC LENGTH SAMPLING METHOD		
1	89.0	97.0
2	97.0	100.0

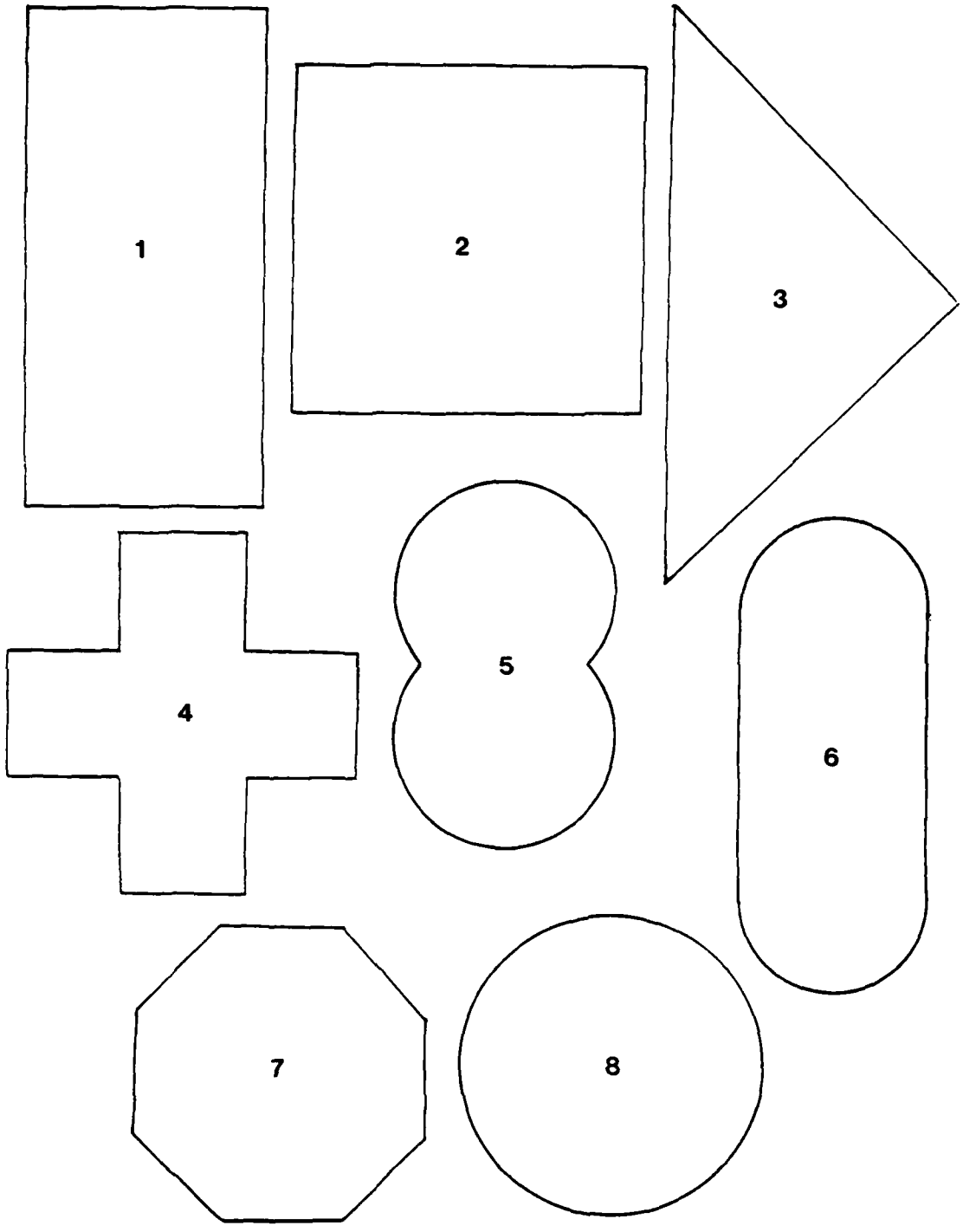


Figure 5.2.3 Geometric Shapes

(reduced to 74%)

Table 5.2.3 Results of the Geometric Shapes Test Using the Optimal K-L Transformation and Both Boundary Sampling Methods

(25 Test Samples = 25 Training Samples)

% Correct Classification

Model Order	Transformed Dimension 1	2	3	4	5	6	7	8	9
1	95.5	98.0							
2	53.5	94.5	98.5						
3	79.5	96.0	97.0	93.5					
4	51.5	92.0	98.5	98.5	95.0				
5	68.0	99.0	99.5	98.0	98.5	98.5			
6	58.5	93.5	98.5	99.0	99.0	99.0	99.5		
7	77.5	95.0	98.0	99.0	98.5	99.0	99.0	99.0	
8	80.5	95.0	97.0	98.5	98.5	99.0	99.0	99.0	99.0
	EQUAL ARC LENGTH SAMPLING METHOD								
1	89.0	98.5							
2	72.5	91.5	93.0						
3	58.5	82.5	83.5	84.5					
4	71.0	86.5	87.5	88.5	89.0				
5	83.0	89.5	85.5	86.5	87.0	87.0			

dimensions of the system using the equal angle sampling method. Note also that the best recognition rate for the equal arc length sampling method occurs at the lowest model order. It is evident upon looking at the top half of Table 5.2.3 that 2 samples (from class 5) of the library data file had AR parameters that deviated too far from the class mean and thus were consistently misclassified. If the test had used all random, new samples, we would probably see an occasional perfect classification score. However, the table does show how the AR parameters can deviate so far from the class mean that even the best transformation technique cannot completely eliminate the effect of this variance.

5.2.4 Multi-Edge Shape Test

This multi-edge shape test is simply an extension of the geometric shapes test. It demonstrates how the pattern recognition system can be used to recognize shapes with multiple edges. For this test we produced 12 shapes with multiple edges by cutting geometrical shapes out of the geometric shapes of the previous section. See Figure 5.2.4 for the representative shapes. The training data for this test was formed from the same samples that were used in the previous test. During classification however, the test data was formed from random, new samples. Thus this is the first test where the test and training data samples are mutually exclusive.

The confusion matrix of one run of this test at model order 5 and transformed dimension 2 is shown in Figure 5.2.5. The confusion matrix shows the number of each of the classifier decisions for a shape, where each shape was submitted to the classifier 20 times. It is observed that some samples were not classified at all, especially samples of shape 1.

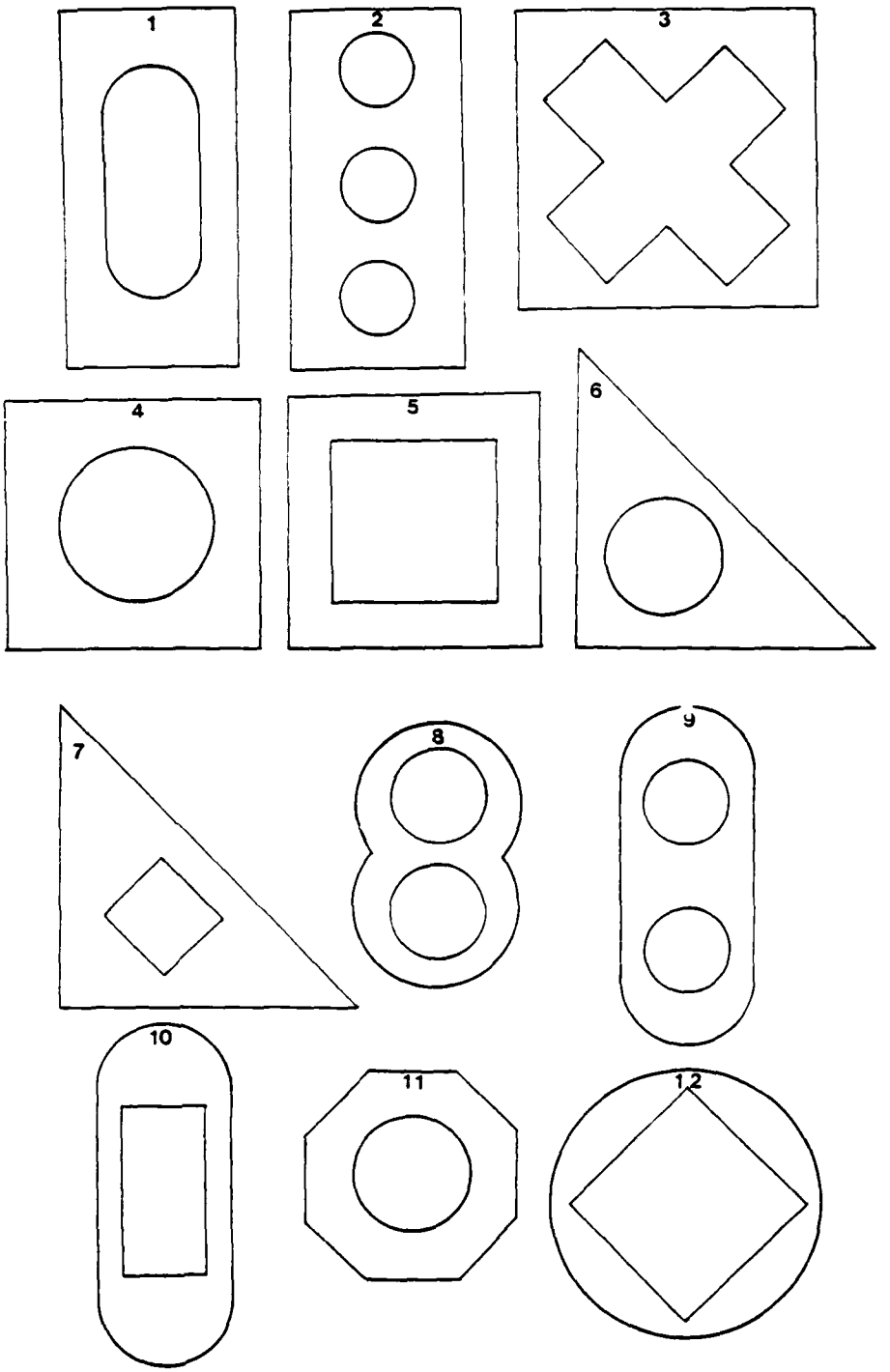


Figure 5.2.4 Multi-Edge Shapes

(reduced to 36%)

Figure 5.2.5 Confusion Matrix of the Multi-Edge Shape Test

(AR Model Order: 5, Transformed Dimension: 2)

(20 Test Samples; 25 Training Samples)

Shape	Classifier Decision											
	1	2	3	4	5	6	7	8	9	10	11	12
1	2											
2		20										
3			9		11							
4				20								
5					20							
6						20						
7							18					
8								15				
9								1	17			
10										20		
11				1							17	
12												20

This problem was caused by the individual edge classifications not matching the actual object vector edge classifications. This apparently high rate of misclassification is due mostly to the greatly reduced size of the interior edges. When the size of an edge becomes very small the pixel quantization problem becomes increasingly significant. The resultant increased noise decreases the overall signal to noise ratio which ultimately degrades system performance. The most common effect of the lowered signal to noise ratio was the classifier confusion of edge 6 with 5 in object 1.

5.2.5 Large Number of Shapes Test

In this test we investigated the performance of the system when the classifier was given a large training set. For this test the sample data files of the industrial, military and geometric shapes were combined to form a single 20 shape data file. The classifier was trained and tested with the samples of this data file using the optimal K-L transformation technique. All the AR parameters in the data file were formed from the time series produced by the equal angle boundary sampling method.

The results of this test are shown in Table 5.2.4. The individual runs of this test started at model order 5 since it was desired that a recognition rate of at least 99% occur for the individual categories before the combination was tested. As can be seen, the classification performance degrades somewhat in comparison to the individual tests. This was expected since the feature vector space becomes more crowded as more shapes are included in the training set.

Table 5.2.4 Results of the Combined Shapes Test Using the Optimal K-L Transformation

% Correct Classification

Model Order	Transformed Dimension 1	2	3	4	5	6	7	8	9
5	53.2	88.0	93.8	95.2	95.4	95.6			
6	57.8	89.2	91.4	95.0	95.4	96.2	97.0		
7	56.2	90.6	94.0	95.4	95.8	97.0	96.6	97.6	
8	44.2	83.8	96.2	90.8	92.6	94.2	94.2	95.0	96.0

5.2.6 Classification Speed

In the previous sections we have investigated the accuracy of the pattern recognition system. Another equally important aspect of a pattern recognition system is classification speed. Usually the classifier is trained off-line, since it is an interactive process, so training speed is desirable but not critical. What is critical and what also determines the possible applications of a pattern recognition system is the speed of classification combined with the accuracy.

In the process of classifying a single edge, the edge is first detected and traced. Then the radius vectors are calculated and the AR parameters are recursively estimated. The AR parameters are then transformed and categorized according to the nearest class mean in the feature space. For multiple edge objects this process is repeated for each edge. Then all the individual edge classifications are ordered and matched to the correct object vector as described in Chapter Four. If the multi-edge object has a unique number of inner boundaries, then it is not necessary to calculate the AR parameters thus reducing the classification time.

The most time consuming tasks of the classification process, using the equal angle boundary sampling method, are ordered below with the most time consuming task first:

1) scanning the entire frame and tracing the boundaries
2) radius vector length calculation
3) AR parameter estimation

The rest of the processes: determining hole ownership, ordering the boundaries, and actual classification do not contribute a significant amount of time to the total classification process. Each of the above tasks are accomplished within a range of times depending on the factors as listed next to each task number below:

1) scan step size, number of edges, total length of each edge
2) number of desired radius vectors, length of the edge (equal angle sampling method)
3) number of resultant radius vectors, and AR model order.

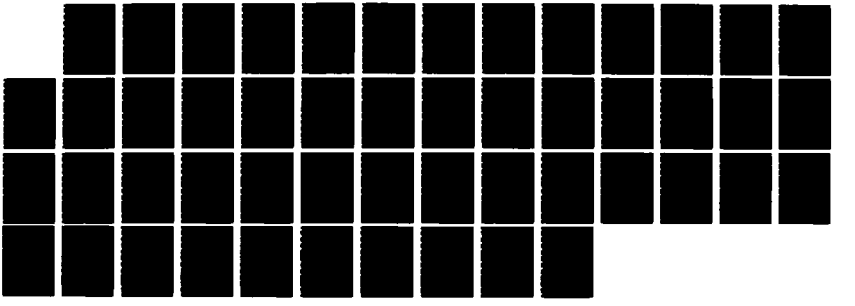
The observed times for scanning the scene and turtling the boundaries range from 0.25 seconds, for no objects in the scene (scan step = 8), to greater than 2.2 seconds for a scene with many objects with multiple edges. Table 2.25 shows the radius vector and AR parameter calculation times for the class 1 (S) shape of the numerical example in the first section of this chapter. The length of the boundary for this shape was 1308 pixels and the scan and turtle time was approximately 0.83 seconds.

Average total times for the classifications of the large shapes of the numerical example are 1.32 seconds using the equal angle sampling method and 1.01 seconds for the equal arc length sampling method. For smaller shapes the total classification time for a single edge is less than one second.

In this chapter we used the programs described in Chapter Four to demonstrate the system theories and the system accuracy. In the first section we present two examples of the feature selection and feature

AD-A171 294 THE CLASSIFICATION OF MULTI-EDGE SHAPES USING AN AUTOREGRESSIVE MODEL AND (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH R D KENNETT DEC 85 2/2

UNCLASSIFIED AFIT/CI/NR-86-132T F/G 6/4 NL



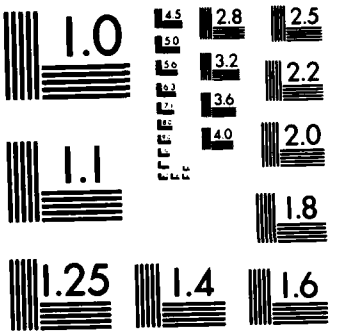


Table 5.2.5 Processor Times for Calculating the Radius Vectors and the AR Parameters (Fractions of a Second)

Model Order	Equal Angle Sampling Method	Equal Arc Length Sampling Method
1	.3398	.0391
2	.3398	.0508
3	.3515	.0560
4	.3789	.0703
5	.3789	.0781
6	.3984	.0781
7	.4023	.0898
8	.4102	.1094
9	.4180	.1211
10	.4336	.1211

vector transformation methods. The second section describes a series of system tests and the resultant classifier performance. In the last part of the tests and results section we summarize the time requirements of this pattern recognition system.

CHAPTER 6

CONCLUSIONS AND SUGGESTED IMPROVEMENTS

In this thesis we have developed a pattern recognition system capable of classifying two dimensional shapes with many edges. Also the system is capable of classifying many shapes in a single scene. Since the problem presented by the existence of many edges in a shape can be considered as a simple extension of the single edge problem, we proceeded to concentrate on improving the single edge classification process. In so doing, we used the work of Dubois [Dubois, 1984] as a starting point and continued by improving some of the existing algorithms and developing a more efficient classification scheme. By applying the theory of the Karhunen-Loeve transformation onto the feature vectors formed from the AR model parameters of an objects boundary, we were able to reduce the dimension of the feature vector space and thus further reduce training data storage space and also classification time.

We have split the contents of this chapter into two sections. The first section provides comments on the work just completed. The second section suggests future work that can be done to enhance the performance of our system.

6.1 Concluding Remarks

On the feature selection techniques:

- The recursive estimation of the AR parameters speeded up classification time without degrading the system performance.

It is noted that most of the time spent in the production of the AR parameters of a given time series is not in the recursion itself, but in calculating the correlation coefficients used by the recursive equations.

- The recursive algorithm did not alter the invariance properties of the AR parameters.

- The performance of the AR model, and thus of the classifier, is dependent on the size of the shape. When the size of the shape as viewed on the monitor decreases, the pixel quantization error increases. Thus the noise on the samples of the time series increases and the AR parameters will deviate further from the class means.

- The equal angle radius vector length calculation method of Dubois was speeded up. The average time was previously 1.68 seconds, this can be compared with the average time of .33 seconds on our system. However, this decreased time is partly at the expense of an increased temporary storage requirement for the boundary pixels.

- The equal arc length boundary sampling technique can be successfully implemented at very low AR model orders. This technique results in an average .31 second reduction in classification time compared with the time of the equal angle sampling method.

- The equal angle boundary sampling method consistently provided better results than the equal arc length sampling method at AR model orders greater than 1.

On the transformation techniques:

- The basic and the optimal Karhunen-Loeve transformation techniques are effective in decorrelating the components of the feature vector. By removing the correlated components corresponding to the smallest eigenvalues of the K-L covariance matrices, the original feature vectors are approximated by vectors of lower dimension.

- The optimal K-L transformation technique proves to be better than the basic K-L transformation when there are more than 2 classes in the feature space. This is because the optimal transformation is based on a between class scatter matrix whereas the basic K-L transformation is based on a within class scatter matrix. In the optimal K-L system the scatter matrix is prewhitened to remove the individual sample variances. This is done to provide maximum use of the discriminatory information contained in the class mean vectors.

- In the industrial shapes test the optimal K-L transformation technique proved to be more effective in separating the feature space than the rotated coordinate system of Dubois. For this test the shapes, which were scaled down versions of the shapes used by Dubois, were correctly classified at AR model orders 3 through 10. Also model orders 5,7,9 required only 2 dimensions

of the transformed space to achieve correct classification. This can be compared with the results of a similar test by Dubois where correct classification was achieved only at model order 5 and maximum dimension of the feature space.

- As in the rotated coordinate system in the work of Dubois, the K-L transformation matrices are based on the eigenvectors of the covariance matrices of the feature vectors. These covariance matrices become singular whenever the number of training samples is less than M+2, where M is the model order. Since we consistently used 25 training samples for all our system tests we did not encounter any singular matrices. However we would like to note that this singular matrix problem can very well occur, especially at high model orders.

- We did not concern ourselves with finding the optimal model order for the tests since the shapes used in the tests were so varied. Generally the best results were obtained with model orders 3 through 10 and with model order 1 for the equal arc length boundary sampling method.

- The permanent storage requirements for the trained classifier in this system consists of an (m+1) x (m+1) dimensioned transformation matrix and c (m+1) transformed class mean feature vectors. As we saw in the examples of Sections 5.1.1 and 5.1.2 where c=2 and m=1, the storage requirement consisted of 8 values. This aspect of this pattern recognition system makes it ideal for the classification of a large set of shapes since the storage requirements for each shape is minimal.

On the original work in this thesis:

- A boundary marking and bookkeeping system was devised to allow multiple shapes with multiple edges in the scene.

- A simple method of determining hole ownership was devised.

- The theory of the Karhunen-Loeve expansion was applied to the vectors formed from the parameters of the AR model of the edges of a shape.

- A special purpose program (MAKLIB) was written to collect the AR parameters at model orders 1 through 10 for any desired edge. This collection of AR parameters was used for classifier training and testing.

- Finally, a complete set of software (TRAIN and CLASSIFY) was developed into a pattern recognition system which not only classifies shapes but also produces size and rotation indicators based on the length and position of the maximum radius vector.

As initially stated, we set out to classify objects with multiple edges. We conclude that we were successful.

6.2 Suggested Improvements

The two most important aspects of a pattern recognition system are speed and accuracy. Hence it is the goal of this section to present suggestions which will increase the classification speed and accuracy.

To increase the classification speed:

- Implement the turtle algorithm in assembly language.

- For multiple edge objects - find the AR parameters of the shape formed by connecting the centroids with straight line segments. This will avoid the time needed to classify each individual edge.

- Segment the scene before scanning to quickly determine the approximate location of an object.

To increase classification accuracy:

- Improve the equal angle boundary sampling technique so that it does not skip over sections of the boundary.

- Determine the optimum AR model order by using Equation 2.2.15e as described in Chapter Two and then use the value as additional information in the classification process.

Also to increase the versatility of the system:

- Modify program TRAIN so that it can train on entire objects without having to isolate the edges.

- Investigate different thresholding techniques to deal with the reflection, shadow, and texture problems of real three dimensional objects.

REFERENCES

REFERENCES

1. Beavers, A.N. and Hubach, R.A.: "Business Outlook: Robot Applications Enhance Vision Sales," High Technology, pg. 61, June 1984.

2. Box, G.E.P. and G.M. Jenkins: Time Series Analysis: Forecasting and Control, San Francisco: Holden-Day, 1976.

3. Brogan, W.L.: Modern Control Theory, New York: Prentice-Hall, 1982.

4. Devijver, P.A. and J. Kittler: Pattern Recognition: A Statistical Approach, London: Prentice-Hall, 1982.

5. Dubois, S.R.: "The Classification of Objects by the Use of Auto-regressive Models of Their Images," Master's Thesis, Dept. Elec. and Comp. Eng., University of New Hampshire, 1984.

6. Duda, R.O. and P.E. Hart: Pattern Classification and Scene Analysis, New York: Wiley, 1973.

7. Durbin, J.: "The Fitting of Time Series Models," Rev. Inst. Int. de Stat., Vol. 28, pp. 233-244, 1960.

8. Edson, D.: "Bin-Picking Robots Punch In," High Technology, pp. 57-60, June 1984.

9. Fu, K.S.: Sequential Methods in Pattern Recognition and Machine Learning, New York: Academic Press, 1968.

10. Granlund, G.H.: "Fourier Preprocessing for Hand Printed Character Recognition," IEEE Transactions on Computers, Vol. 21, pp. 195-201, Feb. 1972.

11. Hu, M.K.: "Visual Pattern Recognition by Moment Invariants," IRE Transactions on Information Theory, Vol. IT-8, pp. 179-187, 1962.

12. Kashyap, R.L. and R. Chellappa: "Stochastic Models for Closed Boundary Analysis: Representation and Reconstruction," IEEE Transactions on Information Theory, Vol. IT-27, pp. 627-637, 1983.

13. Kay, S.M. and S.L. Marple: "Spectrum Analysis - A Modern Perspective," Proceedings of the IEEE, Vol. 69, No. 11, pp.1380-1419, 1981.

14. Kittler, J.: "Mathematical Methods of Feature Selection in Pattern Recognition," International Journal of Man-Machine Studies, Vol. 7, pp. 609-637, 1975.

15. Lanczos, C.: Applied Analysis, Englewood Cliffs, N.J.: Prentice-Hall, 1956.

16. Makhoul, J.: "Linear Prediction: A Tutorial Review, " Proceedings of the IEEE, Vol. 63, No. 4, pp. 561-580, 1975.

17. Pavlidis, T.: "A Review of Algorithms for Shape Analysis," Computer Graphics and Image Processing," Vol. 7, pp. 243-258, 1978.

18. Sebestyen, G.S.: Decision-Making Processes in Pattern Recognition, New York: MacMillan, 1962.

19. Tou, J.T. and R.C. Gonzalez: Pattern Recognition Principles, Reading, Mass.: Addision-Wesley, 1974.

APPENDIX

```
C*************************************************************************
C
C       FILE TRAIN.FOR                      LAST REVISION JULY 1,1985
C                                           AUTHOR: RUTH D. KENNETT
C
C       THIS FILE CONTAINS:
C
C                       PROGRAM TRAIN
C
C       TO LINK TYPE:
C
C       LINK TRAIN,COLLECT,WORK1,RAD1,PARAM,REGIS,KLSEL,MESEL,PREKL,EIGV,-
C            USPKD,SYS$LIBRARY:VIDEOLIB
C
C*************************************************************************
C                       PROGRAM TRAIN
C*************************************************************************
C
C       THIS PROGRAM TRAINS THE CLASSIFIER BY INTERACTIVELY TAKING SAMPLES
C       OF THE CLASSES REPRESENTING THE EDGES OF THE EXPECTED OBJECTS.
C       ONCE THE USER HAS THE DESIRED NUMBER OF SAMPLES OF EACH CLASS, THE
C       SAMPLES ARE THEN SENT TO ONE OF THE DESIRED TRANSFORMATION ROUT-
C       INES: KLSEL,MESEL,PREKL.
C

        INTEGER DEV,IN,OUT,STEP,INRAD,M
        INTEGER DIM,NSAMP,NCLASS,CLASS,BNDCT,SAMP
        INTEGER PCLASS,TOTCLS
        INTEGER INCODE,TRCODE
        INTEGER NEWLUN,OLDLUN
        INTEGER XPIX(2500,10),YPIX(2500,10)
        INTEGER BOUND(10,10)
        REAL    RAD(-2:250,10),ORIENT(10)
        REAL    X(11,25,20),THETA(11,10)
        REAL    TRANS(11,11),T(25,11,20)
        REAL    TA(11,20)
        REAL    LIB(20,25,10,11),ALL(10,11,10)
        CHARACTER*1 INST

        COMMON DEV,IN,OUT,STEP,INRAD,M
        DATA    IN,OUT/5,5/
        DATA    TRANS/121*0.0/

        NEWLUN = 1
        OLDLUN = 2
        OPEN(UNIT=NEWLUN,FILE= 'TRAIN.DAT', STATUS='NEW')
        OPEN(UNIT=OLDLUN,FILE='LIB.DAT',STATUS='OLD')
C
C       SETUP THE IMAGING HARDWARE AND PROMPT FOR DESIRED PARAMETERS
C
        CALL SETUP
        CALL THRESH

        WRITE(OUT,*) 'TYPE 1 - TO TRAIN WITH NEW SAMPLES'
        WRITE(OUT,*) 'TYPE 2 - TO TRAIN WITH SAMPLES FROM LIB.DAT'

        READ(IN,2) INCODE
 2      FORMAT(I2)

        WRITE(OUT,*) 'TYPE 1 - FOR MESEL (MINIMUM ENTROPY)'
```

```
      WRITE(OUT,*) 'TYPE 2 - FOR KLSEL (KARHUNEN-LOEVE)'
      WRITE(OUT,*) 'TYPE 3 - FOR PREKL (OPTIMAL KARHUNEN-LOEVE)'
      READ(IN,3) TRCODE
3     FORMAT(I1)

      DIM = M + 1
      NCLASS = 0
      PRECLS = 0
      IF(INCODE .NE. 2) GOTO 9

      READ(OLDLUN,*) TOTCLS
6     READ(OLDLUN,*) CLASS,SAMP
      IF(CLASS .EQ. 0) GOTO 65
      DO 7 I = 1,10
          READ(OLDLUN,*) (LIB(CLASS,SAMP,I,J),J=1,I+1)
7     CONTINUE
      GOTO 6

9     NSAMP = 0
      GOTO 30

10    IF(INCODE .EQ. 2) GOTO 65
      WRITE(OUT,15) NCLASS,NSAMP
15    FORMAT(1X,'CLASS: ',I2,'  NUMBER OF SAMPLES: ',I2)
      WRITE(OUT,20)
20    FORMAT(1X,'FINISHED WITH SAMPLING? (TYPE Y FOR YES) ',$)
      READ(IN,25) INST
25    FORMAT(A1)
      IF(INST .EQ. 'Y') GOTO 100

      CALL VIEW
30    WRITE(OUT,35)
35    FORMAT(1X,'ENTER CLASS NUMBER: ',$)
      READ(IN,*) CLASS

      IF(CLASS .EQ. PRECLS) GOTO 40
      NCLASS = NCLASS + 1
      PRECLS = CLASS
      NSAMP = 0
C
C     CALCULATE THE AR PARAMTERS OF THE EDGES OF THE TRAINING OBJECTS
C
40    CALL SNAP
      CALL SCAN(BNDCT,BOUND,XPIX,YPIX)
      IF(BNDCT .EQ. 0) GOTO 99
      DO 45 N = 1,BNDCT
          T1 = SECNDS(0.0)
          CALL GETRAD(XPIX(1,N),YPIX(1,N),N,BOUND,RAD)
          CALL PARAM(N,RAD,THETA,ALL)
          DELTA = SECNDS(T1)
          WRITE(DEV,*) 'TIME FOR GETRAD AND PARAM IS: ',DELTA
45    CONTINUE

C
C     FORM THE SAMPLE FEATURE VECTORS FROM THE AR PARAMETERS
C
      DO 60 J = 1,BNDCT
          NSAMP = NSAMP + 1
          DO 50 I = 1,DIM
              X(I,NSAMP,CLASS) = THETA(I,J)
```

```
 50          CONTINUE
 60      CONTINUE
C
C        CALCULATE THE ORIENTATION OF THE EDGES
C
         CALL ROTATE(BNDCT,BOUND,ORIENT)
         CALL OUTPUT(BNDCT,BOUND,RAD,ORIENT)
         GOTO 99

 65      WRITE(OUT,67)
 67      FORMAT(1X,'FINISHED WITH TRAINING? (TYPE Y FOR YES): '$)
         READ(IN,68) INST
 68      FORMAT(A1)
         IF(INST .EQ. 'Y') GOTO 100
         WRITE(OUT,70)
 70      FORMAT(1X,'INPUT CLASS NUMBER: '$)
         READ(IN,80) CLASS
         WRITE(OUT,75)
 75      FORMAT(1X,'INPUT DESIRED NUMBER OF SAMPLES: '$)
         READ(IN,80) NSAMP
 80      FORMAT(I3)

         DO 90 SAMP = 1,NSAMP
             DO 85 I = 1,DIM
                 X(I,SAMP,CLASS) = LIB(CLASS,SAMP,M,I)
 85          CONTINUE
 90      CONTINUE

         IF(CLASS .EQ. PRECLS) GOTO 99
         NCLASS = NCLASS + 1
         PRECLS = CLASS
C
C        LOOP BACK TO GET MORE SAMPLES
C
 99      GOTO 10
C
C        TRANSFORM THE FEATURE VECTORS
C

 100     GOTO(110,120,130) TRCODE

 110     CALL MESEL(X,NSAMP,NCLASS,TRANS,T)
         GOTO 150
 120     CALL KLSEL(X,NSAMP,NCLASS,TRANS,T)
         GOTO 150
 130     CALL PREKL(X,NSAMP,NCLASS,TRANS,T)
C
C     FIND THE AVERAGE OF THE TRANSFORMED SAMPLES
C
 150     DO 180 K = 1,NCLASS
             DO 170 J = 1,DIM
                 SUM = 0.0
                 DO 160 I = 1,NSAMP
                     SUM = SUM + T(I,J,K)
 160             CONTINUE
                 TA(J,K) = SUM / FLOAT(NSAMP)
 170         CONTINUE
 180     CONTINUE
C
C        WRITE THE TRANSFORMED SAMPLE FEATURES TO A TRAIN.DAT FILE
```

```
C
 200    WRITE(NEWLUN,*) DIM

        DO 210 I = 1,DIM
            WRITE(NEWLUN,205) (TRANS(I,J),J=1,DIM)
 205        FORMAT(1X,<DIM>F11.5)
 210    CONTINUE

        WRITE(NEWLUN,230) NSAMP,NCLASS
 230    FORMAT(1X,2I3)

        DO 260 K = 1,NCLASS
            WRITE(NEWLUN,255) (TA(J,K),J=1,DIM)
 255        FORMAT(1X,<DIM>E11.5)
 260    CONTINUE

        IF(DIM-2 .GT. 0) GOTO 290
        WRITE(DEV,285) (I,I=1,DIM)
 285    FORMAT('0','THE AR PARAMETERS OF THE TRAINING DATA'//,
     *  ' CLASS SAMPLE ',7X,<DIM>('M',I2,7X)/)
        GOTO 305

 290    WRITE(DEV,300) (I,I=1,DIM)
 300    FORMAT('0','THE AR PARAMETERS OF THE TRAINING DATA'//,
     *  ' CLASS  SAMPLE ',7X,<DIM-1>('M',I2,7X),'M',I2,3X,/)

 305    DO 350 K = 1,NCLASS
            DO 340 J = 1,NSAMP
                WRITE(DEV,330) K,J,(X(I,J,K),I=1,DIM)
 330            FORMAT(1X,2(I3,5X),<DIM>F10.5)
 340        CONTINUE
 350    CONTINUE

        WRITE(DEV,400) (I,I=1,DIM)
 400    FORMAT('0','THE TRANSFORMED PARAMETERS OF THE TRAINING DATA'//,
     *  ' CLASS  SAMPLE ',7X,<DIM>('T',I2,7X)/)
        DO 450 K = 1,NCLASS
            DO 440 I = 1,NSAMP
                WRITE(DEV,430)K,I,(T(I,J,K),J=1,DIM)
 430            FORMAT(1X,2(I3,5X),<DIM>F10.4)
 440        CONTINUE
 450    CONTINUE

        CLOSE(UNIT=NEWLUN)
        CLOSE(UNIT=OLDLUN)

        STOP
        END
```

```
C*************************************************************************
C
C       FILE WORK1.FOR                  LAST REVISION JULY 1,1985
C
C       ROUTINES CONTAINED IN THIS FILE ARE:
C
C                               SETUP
C                               THRESH
C                               SCAN
C                               GETCEN
C                               INTPLT
C                               OBPLOT
C                               FINPLT
C                               RADPLT
C
C*************************************************************************
C-------------------------------------------------------------------------
C                               SETUP
C-------------------------------------------------------------------------

        SUBROUTINE SETUP
C
C       THIS SUBROUTINE INITIALIZES THE IMAGING SYSTEM HARDWARE AND
C       PROMPTS THE USER FOR DESIRED VALUES
C
        INTEGER DEV,IN,OUT,STEP,INRAD,M
        COMMON  DEV,IN,OUT,STEP,INRAD,M

        CALL SELGRP(1)
        CALL APINIT
        CALL FBINIT
        CALL LUINIT
        CALL SYNC(0)
        CALL VIEW

        WRITE(OUT,40)
 40     FORMAT(1X,'INPUT DESIRED OUTPUT DEVICE: ',$)
        READ(IN,*) DEV

        WRITE(OUT,50)
  50    FORMAT(1X,'INPUT DESIRED SCAN STEP: ',$)
        READ(IN,*) STEP

        WRITE(OUT,100)
 100    FORMAT(1X,'INPUT NUMBER OF RADIUS VECTORS: ',$)
        READ(IN,*) INRAD

        WRITE(OUT,110)
 110    FORMAT(1X,'INPUT THE DESIRED AR MODEL ORDER NUMBER: ',$)
        READ(IN,*) M

        RETURN
        END


C-------------------------------------------------------------------------
C                               THRESH
C-------------------------------------------------------------------------

        SUBROUTINE THRESH
```

```
C
C       THIS SUBROUTINE SETS THE IMAGE THRESHOLD AND FILLS THE FRAME BUFFER
C
        INTEGER DEV,IN,OUT,ANS,YES,THRS,CHTHRS,I
        COMMON  DEV,IN,OUT

        THRS = 120
        INTLO = 0
        INTHI = 255

        CHTHRS = 0
 100    WRITE(OUT,110) THRS
 110    FORMAT(1X,'THRESHOLD = ',I4,' ADD: ',$)
        READ(IN,120) CHTHRS
 120    FORMAT(I3)
        IF(CHTHRS .EQ. 0) GOTO 200
        THRS = THRS + CHTHRS
        GOTO 100

 200    CALL SELLUT(0,0)
        DO 300 I = 0,255
            IF(I .LT. THRS) CALL SETLUT(I,INTLO)
            IF(I .GE. THRS) CALL SETLUT(I,INTHI)
 300    CONTINUE

        RETURN
        END

C-------------------------------------------------------------------------
C                                  SCAN
C-------------------------------------------------------------------------

        SUBROUTINE SCAN(BNDCT,BOUND,X,Y)
C
C       THIS SUBROUTINE SCANS THE FRAME BUFFER FOR ALL OF THE THRESHOLDED
C       EDGES OF THE OBJECTS IN THE IMAGE.  ONCE AN EDGE IS FOUND IT CALLS
C       GETCEN TO CALCULATE THE CENTROID, AND GETRAD TO CALCULATE THE RADIUS
C       VECTOR LENGTHS.
C

        INTEGER RAPIX,RPIXEL,XL,ZP,XN,YN,ZN
        INTEGER BNDCT,BOUND(10,10)
        INTEGER X(2500,10),Y(2500,10),Z,NUMPIX
        INTEGER DEV,IN,OUT,STEP,INRAD
        INTEGER FIRSTX,LASTX,INCX,STOPX
        INTEGER FIRSTY,LASTY
        INTEGER EDGE,BCKGND,OBJ,XTEST
        INTEGER BUF(0:511)
        CHARACTER INST
        LOGICAL YESPLT,GOTONE
        COMMON  DEV,IN,OUT,STEP,INRAD
C
C       FOR WHITE OBJECT ON BLACK BACKGROUND: BCKGND = 0 AND OBJ = 255
C       FOR BLACK OBJECT ON WHITE BACKGROUND: BCKGND = 255 AND OBJ = 0
C
        BCKGND = 0
        OBJ = 255

        EDGE = 255
        FIRSTX = 0
```

```
        LASTX = 511
        INCX = STEP
        STOPX = (LASTX-FIRSTX+1) / STEP
        FIRSTY = 0
        LASTY = 479
        BNDCT = 0

C       WRITE(5,50)
C 50    FORMAT(1X,'TYPE "Y" IF THE OBJECT PLOT IS DESIRED: ',$)
C       READ(5,55) INST
C 55    FORMAT(A1)

        YESPLT = .FALSE.
        IF(INST .NE. 'Y') GOTO 100
        YESPLT = .TRUE.
        CALL INTPLT

 100    T1 = SECNDS(0.0)
        DELTA = 0.0

        DO 500 YN = FIRSTY,LASTY,STEP
C
C           READ EVERY INCX FROM YN LINE IN THE FRAME BUFFER INTO BUFF
C
            CALL RSCAN(YN,FIRSTX,LASTX,INCX,BUF)
            ZP = BUF(0)
            DO 400 XL = 1,STOPX
                ZN = BUF(XL)
C
C               CHECK FOR AN EDGE AND TRACE BACK FROM HOP TO REAL EDGE
C
C               FOR WHITE ON BLACK USE (ZN-ZP) IN NEXT STATEMENT
C               FOR BLACK ON WHITE USE (ZP-ZN) IN NEXT STATEMENT
C

                IF((ZN-ZP) .NE. EDGE) GOTO 300
                XN = (STEP * XL) + FIRSTX
 150            XTEST = XN - 1
                Z = RPIXEL(XTEST,YN)
                IF(Z .GT. BCKGND .AND. Z .LE. BNDCT) GOTO 300
                IF(Z .EQ. BCKGND) GOTO 200
                XN = XTEST
                GOTO 150
C
C               TURTLE AROUND THE EDGE AND CALCULATE THE CENTROID
C
 200            N = BNDCT + 1
                CALL GETCEN(XN,YN,X(1,N),Y(1,N),BNDCT,BOUND,GOTONE)
                IF(.NOT. GOTONE) GOTO 300
C
C               PLOT OUT EDGES
C
                IF(.NOT. YESPLT) GOTO 300
                DELTA = DELTA + SECNDS(T1)
                NUMPIX = BOUND(6,BNDCT)
                CALL OBPLOT(X,Y,NUMPIX)
                T1 = SECNDS(0.0)
 300            ZP = ZN
 400        CONTINUE
 500    CONTINUE
```

```
      DELTA = DELTA + SECNDS(T1)
C     IF(YESPLT) CALL FINPLT
      WRITE(OUT,600) DELTA
 600  FORMAT(1X,'THE SCAN AND TURTLE TIME IS: ',F18.6,' SEC'/)

 1000 RETURN
      END


C-------------------------------------------------------------------------
C                                GETCEN
C-------------------------------------------------------------------------

      SUBROUTINE GETCEN(XS,YS,X,Y,BNDCT,BOUND,GOTONE)
C
C     THIS SUBROUTINE CALULATES THE CENTROID OF THE BOUNDARY.  THE EDGE
C     IS TRACED USING A TURTLE ALGORITHM.
C
      INTEGER XS,YS,BNDCT,BOUND(10,10)
      INTEGER XTEST
      INTEGER RPIXEL,RAPIX,XT,YT,Z,BNDINT
      INTEGER XREG,YREG,NUMPIX,XC,YC
      INTEGER X(2500),Y(2500)
      INTEGER N,TOL,BCKGND
      LOGICAL OBJECT,RIGHT,UP,GOTONE


      TOL = 25
      BCKGND = 0
      N = BNDCT + 1
      BNDINT = N

      CALL WPIXEL(XS,YS,BNDINT)
      XREG = XS
      YREG = YS
      NUMPIX = 1
      X(NUMPIX) = XS
      Y(NUMPIX) = YS
      XT = XS - 1
      YT = YS - 1
      UP = .TRUE.

 190  Z = RPIXEL(XT,YT)
      IF(Z .EQ. BCKGND) GOTO 250
      IF((XT.EQ.XS) .AND. (YT.EQ.YS) .AND. (NUMPIX.GT.10)) GOTO 500
      IF(Z .LT. BNDINT) GOTO 500
 200  OBJECT = .TRUE.
      IF((XT.EQ.X(NUMPIX)).AND.(YT.EQ.Y(NUMPIX))) GOTO 260
      XREG = XREG + XT
      YREG = YREG + YT
      NUMPIX = NUMPIX + 1
      X(NUMPIX) = XT
      Y(NUMPIX) = YT
      CALL WPIXEL(XT,YT,BNDINT)
      GOTO 260
 250  OBJECT = .FALSE.
C
C     MOVE UP OR DOWN
C
```

```
 260    IF((UP.AND.OBJECT).OR.((.NOT.UP).AND.(.NOT.OBJECT))) GOTO 275
        XT = XT + 1
        RIGHT = .TRUE.
        GOTO 290
 275    XT = XT - 1
        RIGHT = .FALSE.
C
C       CHECK FOR OBJECT PIXEL
C
 290    Z = RPIXEL(XT,YT)
        IF(Z .EQ. BCKGND) GOTO 350
        IF((XT.EQ.XS) .AND. (YT.EQ.YS) .AND.(NUMPIX.GT.10)) GOTO 500
        IF(Z .LT. BNDINT) GOTO 500
 300    OBJECT = .TRUE.
        IF((XT.EQ.X(NUMPIX)) .AND. (YT.EQ.Y(NUMPIX))) GOTO 360
        XREG = XREG + XT
        YREG = YREG + YT
        NUMPIX = NUMPIX + 1
        X(NUMPIX) = XT
        Y(NUMPIX) = YT
        CALL WPIXEL(XT,YT,BNDINT)
        GOTO 360
 350    OBJECT = .FALSE.
C
C       MOVE UP OR DOWN
C
 360    IF((RIGHT.AND.OBJECT).OR.((.NOT.RIGHT).AND.(.NOT.OBJECT)))
     *  GOTO 375
        YT = YT + 1
        UP = .FALSE.
        GOTO 390
 375    YT = YT - 1
        UP = .TRUE.
 390    GOTO 190
C
C       CALCULATE THE XC,YC OF THE CENTROID PIXEL
C
 500    IF(NUMPIX .GT. TOL) GOTO 600
        GOTONE = .FALSE.
        GOTO 1000

 600    BNDCT = BNDCT + 1
        GOTONE = .TRUE.
        XC = XREG / NUMPIX
        YC = YREG / NUMPIX

        X(NUMPIX+1) = XS
        Y(NUMPIX+1) = YS
        BOUND(1,N) = BNDINT
        BOUND(2,N) = XS
        BOUND(3,N) = YS
        BOUND(4,N) = XC
        BOUND(5,N) = YC
        BOUND(6,N) = NUMPIX

 1000   RETURN
        END

C-------------------------------------------------------------------------
C                                  INTPLT
```

```
C-----------------------------------------------------------------------

      SUBROUTINE INITPLT
C
C     THIS SUBROUTINE INITIALIZES THE PLOTTER.
C
      INTEGER LUN
      COMMON /DEV_DAT/LUN

      LUN = 5
      CALL TXTCLR
      CALL PLTINT
      CALL PLTCLR
C
C     PLOT THE FRAME
C
      CALL PLTCUR(0,0)
      CALL PLTVEC(511,0)
      CALL PLTVEC(511,479)
      CALL PLTVEC(0,479)
      CALL PLTVEC(0,0)

      RETURN
      END


C-----------------------------------------------------------------------
C                              OBPLOT
C-----------------------------------------------------------------------

      SUBROUTINE OBPLOT(X,Y,NUMPIX)
C
C     THIS SUBROUTINE PLOTS OUT THE POINTS COMPRISING THE EDGE COORDIN-
C     ATES HELD IN ARRAY X AND ARRAY Y.
C
      INTEGER X(3000),Y(3000),NUMPIX


      DO 100 I = 1,NUMPIX
          IX = X(I)
          IY = Y(I)
          CALL PLTPNT(IX,IY)
 100  CONTINUE

      RETURN
      END


C-----------------------------------------------------------------------
C                              FINPLT
C-----------------------------------------------------------------------

      SUBROUTINE FINPLT
C
C     THIS SUBROUTINE PRINTS OUT A HARD COPY OF THE PLOT IF THE USER SO
C     DESIRES IT.
C
      CHARACTER INST

      WRITE(5,50)
```

```
 50     FORMAT(1X,'TYPE "Y" IF HARD COPY IS DESIRED: ',$)
        READ(5,55) INST
 55     FORMAT(A1)

        IF(INST .NE. 'Y') GOTO 100
        CALL HARDCOPY
 100    CALL PLTCLR
        CALL TXTCLR
        CALL PLTOFF

        RETURN
        END

C-------------------------------------------------------------------------
C                                  RADPLT
C-------------------------------------------------------------------------

        SUBROUTINE RADPLT(RAD,EDGE)

        INTEGER EDGE,NUMRAD
        REAL    RAD(-2:250,10)

        N = EDGE
        MAXY = 479
        RNUMRAD = RAD(-2,N)
        NUMRAD = INT(RAD(-2,N))
        INC = 512 / NUMRAD
        IXB = 0
        IYB = MAXY

        DO 100 I = 1,NUMRAD
            IXB = INT(FLOAT(I)*511.0/RNUMRAD)
            IYT = MAXY - (INT(RAD(I,N)) * 2)
            CALL PLTCUR(IXB,IYT)
            CALL PLTVEC(IXB,IYB)
 100    CONTINUE

        RETURN
        END
```

```
C*************************************************************************
C
C       FILE RAD1.FOR                       LAST REVISION JULY 1, 1985
C
C       THIS FILE CONTAINS THE SUBROUTINE:
C
C                                   GETRAD
C
C*************************************************************************
C-------------------------------------------------------------------------
C                                   GETRAD
C-------------------------------------------------------------------------

        SUBROUTINE GETRAD(X,Y,EDGE,BOUND,RAD)

C       THIS SUBROUTINE FINDS AND CALCULATES THE RADIUS VECTORS OF
C       AN OBJECT EDGE.  IT USES AN EQUI-ANGLE ALGORITHM SIMILAR
C       TO DUBOIS [1984]. THIS SUBROUTINE READS THE EDGE PIXEL LOCATIONS
C       WHICH ARE TEMPORARILY STORED IN THE X AND Y ARRAYS.
C
        INTEGER X(2500),Y(2500),BOUND(10,10),EDGE
        INTEGER DEV,IN,OUT,I,N,M,XC,YC,NUMPIX
        INTEGER INRAD,NSLOPE,LOWVEC,HIVEC,SEC
        INTEGER XMAX,YMAX
        REAL    RAD(-2:250,10),SLOPE(0:90),PI,PHI
        REAL    VEC,XDIF,YDIF,YTRY1,YTRY2,YDR1,YDR2
        REAL    YD1,YD2,MAXRAD
        LOGICAL START,STOP
        COMMON  DEV,IN,OUT,STEP,INRAD
        DATA    PI/3.141592654/

        N = EDGE
        XC = BOUND(4,N)
        YC = BOUND(5,N)
        NUMPIX = BOUND(6,N)
        NSLOPE = INRAD/4 - 1
        PHI = (2.0 * PI) / FLOAT(INRAD)

        DO 50 I = 0,NSLOPE
           SLOPE(I) = TAN(PHI * FLOAT(I)) * (5.0/4.0)
 50     CONTINUE
        SLOPE(NSLOPE + 1) = 1250.0

        STOP = .FALSE.
        START = .TRUE.
        I = 0
        PREVEC = 0
        SEC = 0
        M = 0
        MAXRAD = 0.0
C
C    CALCULATE THE SLOPE OF THE LINE FROM THE EDGE PIXEL TO THE CENTROID
C
 100    I = I + 1
        IF(I .GE. NUMPIX) STOP = .TRUE.
        XDIF = FLOAT(X(I) - XC)
        YDIF = FLOAT(Y(I) - YC)
        IF(XDIF .EQ. 0.0) GOTO 700
        VEC = ABS(YDIF/XDIF)
        IF(ABS(VEC - PREVEC) .LE. .0001) GOTO 100
```

```
        LOWVEC = SEC
        PREVEC = VEC
C
C     FIND OUT WHICH SECTOR WE ARE IN
C
 110    IF(VEC .GE. SLOPE(LOWVEC) .AND. VEC .LE. SLOPE(LOWVEC+1))
     *  GOTO 120

        IF(VEC .LE. SLOPE(LOWVEC) .AND. VEC .GE. SLOPE(LOWVEC-1))
     *  GOTO 140

        IF(LOWVEC .GE. NSLOPE) GOTO 115
        LOWVEC = LOWVEC + 1
        GOTO 110
 115    LOWVEC = 1
        GOTO 110

 120    HIVEC = LOWVEC + 1
        GOTO 200
 140    HIVEC = LOWVEC - 1
C
C     CALCULATE THE REFERENCE POSITION BETWEEN LOVEC AND HIVEC
C
 200    YTRY1 = ABS(XDIF) * SLOPE(LOWVEC)
        YTRY2 = ABS(XDIF) * SLOPE(HIVEC)
        YDR1 = ABS(YDIF) - YTRY1
        YDR2 = ABS(YDIF) - YTRY2
C
C     LOOK AT THE NEXT POSITION ALONG THE EDGE
C
300     I = I + 1
        IF(I .GE. NUMPIX) STOP = .TRUE.
        XDIF = FLOAT(X(I) - XC)
        YDIF = FLOAT(Y(I) -YC)
        IF(XDIF .EQ. 0.0) GOTO 700
        GOTO 800

 700    HIVEC = NSLOPE + 1
        LOWVEC = NSLOPE
        GOTO 1000
C
C     CALCULATE THE NEW POSITION WITH RESPECT TO HIVEC AND LOWVEC
C
 800    YTRY1 = ABS(XDIF) * SLOPE(LOWVEC)
        YTRY2 = ABS(XDIF) * SLOPE(HIVEC)
        YD1 = ABS(YDIF) - YTRY1
        YD2 = ABS(YDIF) - YTRY2
C
C     CHECK IF WE HAVE CROSSED LOVEC
C
        IF((YD1 .GE. 0.0 .AND. YDR1 .LE. 0.0) .OR. (YD1 .LE. 0.0
     *  .AND. YDR1 .GE. 0.0)) GOTO 900
C
C     CHECK IF WE HAVE CROSSED HIVEC
C
        IF((YD2 .GE. 0.0 .AND. YDR2 .LE. 0.0) .OR. (YD2 .LE. 0.0
     *  .AND. YDR2 .GE. 0.0)) GOTO 1000
C
C     HAVEN'T CROSSED LOVEC OR HIVEC YET, SO GET THE NEXT EDGE PIXEL
C
```

```
        YDR1 = YD1
        YDR2 = YD2
        IF(STOP) GOTO 2000
        GOTO 300
C
C     YEH, MADE IT ACROSS A SLOPE MARKER (LOVEC OR HIVEC)
C     NOW WE CAN CALCULATE THE RADIUS VECTOR LENGTH
C
C     WE CROSSED LOVEC
C
 900    IF(LOWVEC .EQ. SEC) GOTO 100
        M = M + 1
        RAD(M,N) = SQRT((XDIF*1.25)**2 + YDIF**2)
        SEC = LOWVEC
        IF(STOP) GOTO 2000
        GOTO 1200
C
C     WE CROSSED HIVEC
C
 1000   IF(HIVEC .EQ. SEC) GOTO 100
        M = M + 1
        RAD(M,N) = SQRT((XDIF*1.25)**2 + YDIF**2)
        SEC = HIVEC
        IF (STOP) GOTO 2000

 1200   IF(RAD(M,N) .LE. MAXRAD) GOTO 100
        MAXRAD = RAD(M,N)
        XMAX = X(I)
        YMAX = Y(I)
        GOTO 100

 2000   RAD(-2,N) = FLOAT(M)
        RAD(-1,N) = MAXRAD
        BOUND(9,N) = XMAX
        BOUND(10,N) = YMAX
C       CALL FINPLT

C       CALL INTPLT
C       CALL RADPLT(RAD,EDGE)
C       CALL FINPLT

        RETURN
        END
```

```
C*********************************************************************
C
C       FILE RAD2.FOR                       LAST REVISION JULY 26,1985
C
C       SUBROUTINES CONTAINED IN THE FILE:
C
C                                 GETRAD
C
C*********************************************************************

        SUBROUTINE GETRAD(X,Y,BNDCT,BOUND,RAD)
C
C       THIS SUBROUTINE CALCULATES THE OBJECT EDGE RADIUS LENGTHS
C       USING THE EQUAL ARC LENGTH ALGORITHM .
C
        INTEGER DEV,IN,OUT,STEP,INRAD
        INTEGER X(2500),Y(2500),BNDCT,BOUND(10,10)
        INTEGER XC,YC,NUMPIX,INC
        INTEGER XMAX,YMAX
        REAL    RAD(-2:300,10),XDIF,YDIF,MAXRAD

        COMMON  DEV,IN,OUT,STEP,INRAD

        N = BNDCT
        XC = BOUND(4,N)
        YC = BOUND(5,N)
        NUMPIX = BOUND(6,N)

        MAXRAD = 0.0

        DO 200 I = 1,INRAD
            INC = I * NUMPIX / INRAD
            XDIF = FLOAT(X(INC) - XC)
            YDIF = FLOAT(Y(INC) - YC)
            RAD(I,N) = SQRT((XDIF*1.25)**2 + YDIF**2)
C           CALL PLTCUR(XC,YC)
C           CALL PLTVEC(X(INC),Y(INC))
            IF(RAD(I,N) .LT. MAXRAD) GOTO 200
            MAXRAD = RAD(I,N)
            XMAX = X(INC)
            YMAX = Y(INC)
 200    CONTINUE

        RAD(-2,N) = FLOAT(INRAD)
        RAD(-1,N) = MAXRAD
        BOUND(9,N) = XMAX
        BOUND(10,N) = YMAX
C       CALL FINPLT

C       CALL INTPLT
C       CALL RADPLT(RAD,N)
C       CALL FINPLT
        RETURN
        END
```

```
C*************************************************************************
C
C       FILE PARAM.FOR      LAST REVISION  JULY 1,1985
C
C       THIS FILE CONTAINS:
C
C                               TEST ROUTINE FOR PARAM
C                               PARAM
C
C*************************************************************************
C
C       THIS IS A TEST ROUTINE USED TO CHECK THE AR COEFF-
C       ICIENTS AND THE SUBROUTINE PARAM WHICH CALCULATES
C       THE AUTOCORRELATION AND THE AUTOREGRESSIVE COEFFICIENTS
C       OF THE TIME SERIES.
C
C       THE TEST ROUTINE
C
C       INTEGER DEV
C       REAL    RAD(-2:300,10),ALPH(11,10),RX(0:10,10)
C       COMMON DEV
C       DEV = 5
C
C       N = 1
C       RAD(1,N) = 1.
C       RAD(2,N) = 1.44
C       RAD(3,N) = 1.44*RAD(2,N) - 1.26
C       DO 10 J = 4,40
C           RAD(J,N) = 1.44*RAD(J-1,N) - 1.26*RAD(J-2,N) + .81*RAD(J-3,N)
C 10    CONTINUE
C
C       RAD(-2,N) = 40.
C
C       CALL PARAM(N,RAD,ALPH,RX)
C       STOP
C       END

C-------------------------------------------------------------------------
C                                PARAM
C-------------------------------------------------------------------------

        SUBROUTINE PARAM(EDGE,RAD,THETA,ALL)
C
C       THIS SUBROUTINE CALCULATES THE FEATURE VECTOR PARAMETERS OF THE
C       EDGES OF THE OBJECT.  THE PARAMETERS ARE THE AR COEFFICIENTS AND
C       THE TERM ALPHA/SQRT(BETA).
C
        INTEGER DEV,IN,OUT,STEP,INRAD,M
        INTEGER EDGE,NUMRAD
        REAL    SUM,SUM1,SUM2
        REAL    ALPHA,BETA,ALBET
        REAL    RAD(-2:250,10),THETA(11,10),RX(0:10,10)
        REAL    ALL(10,11,10)
        REAL    RXN(0:10,10)
        REAL    SIGMAS(10),A(10,10)
        COMMON  DEV,IN,OUT,STEP,INRAD,M

        N = EDGE
        DO 30 J = 1,10
            DO 20 I = 1,10
```

```
                A(I,J) = 0.0
 20         CONTINUE
 30     CONTINUE

        NUMRAD = INT(RAD(-2,N))
C
C       CALCULATE THE MEAN
C
        SUM = 0.0
        DO 50 I = 1,NUMRAD
            SUM = SUM + RAD(I,N)
 50     CONTINUE
        MEAN = SUM / FLOAT(NUMRAD)
C
C       CALCULATE THE AUTOCORRELATION COEFFICIENTS
C
        DO 100 I = 0,M
            SUM = 0.0
            DO 75 J = 1,NUMRAD
                K = J + I
                IF(K .GT. NUMRAD) K = K - NUMRAD
                SUM = SUM + ((RAD(J,N) - MEAN) * (RAD(K,N) - MEAN))
 75         CONTINUE
            RX(I,N) = SUM / FLOAT(NUMRAD)
            RXN(I,N) = RX(I,N) / RX(0,N)
 100    CONTINUE
C
C       CALCULATE THE AUTOREGRESSIVE COEFFICIENTS USING THE DURBIN
C       RECURSIVE ALGORITHM
C
        A(1,1) = -RX(1,N) / RX(0,N)
        SIGMAS(1) = (1.0 - (A(1,1) * A(1,1))) * RX(0,N)

        DO 300 K = 2,M
            SUM = 0.0
            DO 250 J = 1,K - 1
                SUM = SUM + (A(K-1,J) * RX(K-J,N))
 250        CONTINUE

            A(K,K) = -(RX(K,N) + SUM) / SIGMAS(K-1)

            DO 275 I = 1,K-1
                A(K,I) = A(K-1,I) + (A(K,K) * A(K-1,K-I))
 275        CONTINUE

            SIGMAS(K) = (1.0-(A(K,K) * A(K,K))) * SIGMAS(K-1)
 300    CONTINUE
C
C       FORM THE FEATURE VECTOR FOR THE EDGE FROM THE LAST COLUMN
C       OF THE MATRIX A.  THIS COLUMN CORRESPONDS TO THE DESIRED
C       AR MODEL ORDER NUMBER.  ALSO FORM THE SUM OF THE PARAMETERS
C       NEEDED IN THE CALCULATION OF ALPHA.
C
        SUM = 0.0
        DO 400 K = 1,M
            THETA(K,N) = A(M,K)
            SUM = SUM + THETA(K,N)
 400    CONTINUE
        ALPHA = MEAN * (1.0 - SUM)
C
```

```
C       CALCULATE BETA
C
        BETA = 0.0
        DO 420 I = 1,NUMRAD
            SUM = 0.0
            DO 410 J = 1,M
                K = I + J
                IF(K .GT. NUMRAD) K = K - NUMRAD
                SUM = SUM + THETA(J,N) * RAD(K,N)
 410        CONTINUE

            SUM1 = RAD(I,N) - ALPHA - SUM
            BETA = BETA + SUM1**2
 420    CONTINUE

        BETA = BETA / FLOAT(NUMRAD)
C       BETA = SIGMAS(M)
        ALBET = ALPHA / SQRT(BETA)
        THETA(M+1,N) = ALBET


C           WRITE(DEV,425) (I,(A(K,I),K=1,M),I=1,M)
C 425       FORMAT(1X,I3,3X,<M>F8.3)

C           WRITE(DEV,450) (J,SIGMAS(J),J=1,M)
C 450       FORMAT(1X,I3,E18.6)


C       WRITE(DEV,700)
C 700   FORMAT(1X,/' THE AUTOCORRELATION COEFFICIENTS (PARAM): '/)
C       WRITE(DEV,705) (J,(RX(J,N),N=1,BNDCT),J = 0,M)

C       WRITE(DEV,702)
C 702   FORMAT(1X,/' THE NORMALIZED AUTOCORRELATION COEFS (PARAM): '/)
C       WRITE(DEV,705) (J,(RXN(J,N),N=1,BNDCT),J=0,M)
C 705   FORMAT(1X,I3,3X,<BNDCT>F10.5)


C       WRITE(DEV,710)
C 710   FORMAT(1X,/' THE AUTOREGRESSIVE COEFFICIENTS (PARAM): '/)
C       WRITE(DEV,715) (J,(THETA(J,N),N=1,BNDCT),J = 1,M+1)
C 715   FORMAT(1X,I3,3X,<BNDCT>F10.5)


        RETURN
        END
```

```
C*****************************************************************************
C
C       FILE KLSEL.FOR          LAST REVISION   JULY 4, 1985
C
C       THIS FILE CONTAINS SUBROUTINE:
C
C                               KLSEL
C
C*****************************************************************************
C-----------------------------------------------------------------------------
C                               KLSEL
C-----------------------------------------------------------------------------
C
        SUBROUTINE KLSEL(X,NSAMP,NCLASS,TRANS,Y)
C
C       THIS SUBROUTINE CALCULATES THE KARHUNEN-LOEVE EXPANSION
C       TRANSFORM MATRIX AND TRANSFORMS THE SAMPLES OF EACH
C       PATTERN CLASS OF THE TRAINING SET.
C

        INTEGER DEV,IN,OUT,STEP,INRAD,M
        INTEGER DIM,JOBN,IZ,IER
        INTEGER NCLASS,NSAMP,MAXI
        REAL    A(100),WK(100),D(11),Z(100)
        REAL    X(11,25,10),XMN(11,25,10),R(11,11,10),C(11,11)
        REAL    MEAN(11,10),MMT(11,11,10)
        REAL    SCALE,MAXE
        REAL    TRANS(11,11),Y(25,11,10)

        COMMON  DEV,IN,OUT,STEP,INRAD,M

        DATA    R/1210*0.0/
        DATA    C/121*0.0/

        T1 = SECNDS(0.00)
        DIM = M + 1
C
C       CALCULATE THE MEAN SAMPLE FOR EACH CLASS
C
        SCALE = 1.0 / FLOAT(NSAMP)
        DO 50 N = 1,NCLASS
            DO 40 I = 1,DIM
                SUM = 0.0
                DO 30 J = 1,NSAMP
                   SUM = SUM + X(I,J,N)
 30             CONTINUE
                MEAN(I,N) = SUM * SCALE
 40         CONTINUE
 50     CONTINUE
C
C       SUBTRACT THE MEAN FROM EACH SAMPLE IN THE CLASS
C
        DO 100 N = 1,NCLASS
            DO 75 K = 1,NSAMP
                DO 60 I = 1,DIM
                    XMN(I,K,N) = X(I,K,N) - MEAN(I,N)
 60             CONTINUE
 75         CONTINUE
100     CONTINUE
```

```
C
C       FORM THE COVARIANCE MATRIX OF EACH CLASS
C
        DO 200 N = 1,NCLASS
            DO 175 K = 1,NSAMP
                DO 150 I = 1,DIM
                    DO 125 J = 1,DIM
                        R(I,J,N) = R(I,J,N) + XMN(I,K,N) * XMN(J,K,N)
 125                CONTINUE
 150            CONTINUE
 175        CONTINUE
 200    CONTINUE

C
C       CALCULATE THE AVERAGE COVARIANCE MATRIX
C
        SCALE = 1.0 / FLOAT(NCLASS * NSAMP)

        DO 300 I = 1,DIM
            DO 275 J = 1,DIM
                DO 250 N = 1,NCLASS
                    C(I,J) = C(I,J) + (SCALE * R(I,J,N))
 250            CONTINUE
 275        CONTINUE
 300    CONTINUE

C       DO 325 I = 1,DIM
C           WRITE(DEV,*) 'C ',I,(C(I,J),J = 1,DIM)
C 325   CONTINUE
C
C       PUT THE LOWER TRAINGLE OF MATRIX C INTO ARRAY A (SYMMETRIC STORAGE
C       MODE)
C
        K = 1
        DO 340 I = 1,DIM
            DO 335 J = 1,I
                A(K) = C(I,J)
C               WRITE(DEV,*) 'A ',A(K)
                K = K + 1
 335        CONTINUE
 340    CONTINUE
C
C       CALCULATE THE EIGENVALUES AND THE EIGENVECTORS
C
        JOBN = 2
        IZ = DIM

        CALL EIGRS(A,DIM,JOBN,D,Z,IZ,WK,IER)

C       K = 1
C       DO 350 I = 1,DIM
C           WRITE(DEV,*)
C           WRITE(DEV,*) 'EIGENVALUE: ', D(I)
C           WRITE(DEV,*)
C           DO 348 N = 1,DIM
C               WRITE(DEV,*) 'EIGENVECTOR: ',Z(K)
C               K = K + 1
C 348       CONTINUE
C 350   CONTINUE
```

```
C       WRITE(5,*) 'WK(1) ',WK(1)
C
C
C       FORM THE COLUMNS OF THE TRANSFORMATION MATRIX FROM THE EIGENVECTORS
C       CORRESPONDING TO THE LARGEST EIGENVALUES.
C
        IMAX = DIM * (DIM - 1)
        DO 525 J = 1,DIM
            DO 515 I = 1,DIM
                TRANS(I,J) = Z(IMAX+I)
 515        CONTINUE
            IMAX = IMAX - DIM
 525    CONTINUE
C       WRITE(DEV,*) 'THE TRANSFORMATION MATRIX'
C       WRITE(DEV,527) ((TRANS(I,J),J=1,DIM),I=1,DIM)
C 527   FORMAT(1X,<DIM>E15.6)

C
C       TRANSFORM THE ORIGINAL SAMPLES OF EACH CLASS
C
        DO 600 K = 1,NCLASS
            DO 575 J = 1,NSAMP
                DO 550 N = 1,DIM
                    Y(J,N,K) = 0.0
                    DO 530 I = 1,DIM
                        Y(J,N,K) = Y(J,N,K) + (TRANS(I,N) * X(I,J,K))
 530                CONTINUE
 550            CONTINUE
 575        CONTINUE
 600    CONTINUE

        DELTA = SECNDS(T1)
        WRITE(DEV,*) 'KLSEL EXECUTION TIME IS: ',DELTA
C       WRITE(DEV,*) 'THE TRANSFORMED SAMPLES'
C       WRITE(DEV,700) (((Y(J,N,K),N=1,DIM),J = 1,NSAMP),K = 1,NCLASS)
C 700   FORMAT(1X,<DIM>E15.6)

        RETURN
        END
```

```
C*************************************************************************
C                                MESEL
C*************************************************************************

        SUBROUTINE MESEL IS EXACTLY THE SAME AS SUBROUTINE KLSEL EXCEPT
        THAT THE ORDERING OF THE EIGENVECTORS IN THE RESULTING TRANS-
        FORMATION MATRIX IS REVERSED. TO AVOID REPETITION, THE LISTING
        HAS BEEN OMITTED.
C-------------------------------------------------------------------------
C-------------------------------------------------------------------------
```

```
C*************************************************************************
C
C       FILE PREKL.FOR                    LAST REVISION JULY 6,1985
C
C       THIS FILE CONTAINS SUBROUTINE:
C
C                                 PREKL
C
C
C*************************************************************************
C-------------------------------------------------------------------------
C                                 PREKL
C-------------------------------------------------------------------------

        SUBROUTINE PREKL(X,NSAMP,NCLASS,TRANS,Y)

        INTEGER DEV,IN,OUT,STEP,INRAD,M
        INTEGER DIM,JOBN,IZ,IER
        INTEGER NCLASS,NSAMP
        REAL    SCALE,SUM
        REAL    X(11,25,20),TRANS(11,11),Y(25,11,20)
        REAL    MEAN(11,20),XMN(11,25,20),AVEMEAN(11)
        REAL    SW(11,11),R(11,11,20)
        REAL    A(100),Z(121),D(11),WK(150)
        REAL    B(11,11)
        REAL    SBSUB(11,11),SBPRIME(11,11),SB(11,11)

        COMMON  DEV,IN,OUT,STEP,INRAD,M

        DATA    R/2420*0.0/
        DATA    SW/121*0.0/
        DATA    SB/121*0.0/

        T1 = SECNDS(0.00)
        DIM = M + 1
        SCALE = 1.0 / FLOAT(NSAMP)
C
C       CALCULATE THE MEAN
C
        DO 100 N = 1,NCLASS
            DO 75 I = 1,DIM
                SUM = 0.0
                DO 50 J = 1,NSAMP
                    SUM = SUM + X(I,J,N)
 50             CONTINUE
                MEAN(I,N) = SUM * SCALE
 75         CONTINUE
 100    CONTINUE
C
C       CALCULATE THE MEAN VECTOR * MEAN VECTOR (TRANSPOSED) FOR EACH
C       CLASS
C
        DO 200 N = 1,NCLASS
            DO 175 K = 1,NSAMP
                DO 150 I = 1,DIM
                    XMN(I,K,N) = X(I,K,N) - MEAN(I,N)
 150            CONTINUE
 175        CONTINUE
 200    CONTINUE
```

```
C
C       CALCULATE PART OF THE COVARIANCE MATRIX FOR EACH CLASS
C
        DO 300 N = 1,NCLASS
            DO 275 K = 1,NSAMP
                DO 250 I = 1,DIM
                    DO 225 J = 1,DIM
                        R(I,J,N) = R(I,J,N) + XMN(I,K,N) * XMN(J,K,N)
 225                CONTINUE
 250            CONTINUE
 275        CONTINUE
 300    CONTINUE

C
C       FORM THE AVERAGE CLASS COVARIANCE MATRIX
C
        SCALE = 1.0 / FLOAT(NSAMP * NCLASS)

        DO 400 I = 1,DIM
            DO 375 J = 1,DIM
                DO 350 N = 1,NCLASS
                    SW(I,J) = SW(I,J) + (SCALE * R(I,J,N))
 350            CONTINUE
 375        CONTINUE
 400    CONTINUE
C       DO 410 I = 1,DIM
C           WRITE(DEV,*) (SW(I,J),J = 1,DIM)
C 410   CONTINUE
C
C       PUT THE LOWER TRIANGLE OF MATRIX SW INTO ARRAY A (FOR SYMMETRIC
C       STORAGE MODE)
C
        K = 1
        DO 450 I = 1,DIM
            DO 425 J = 1,I
                A(K) = SW(I,J)
                K = K + 1
 425        CONTINUE
 450    CONTINUE
C
C       CALCULATE THE EIGENVALUES AND EIGENVECTORS OF SW (IN ARRAY A)
C
        JOBN = 2
        IZ = DIM
        CALL EIGRS(A,DIM,JOBN,D,Z,IZ,WK,IER)
C
C       FORM MATRIX B FROM THE EIGENVECTORS / SQRT(EIGENVALUE)
C
        IMAX = DIM * (DIM - 1)
        DO 525 J = 1,DIM
            K = DIM - J + 1
C           WRITE(DEV,*)
C           WRITE(DEV,*) 'EIGENVALUE: ',D(K)
C           WRITE(DEV,*)
            DO 510 I = 1,DIM
C               WRITE(DEV,*) 'EIGENVECTOR: ',Z(IMAX+I)
                B(I,J) = Z(IMAX+I) / SQRT(D(K))
C               WRITE(DEV,*) 'B(I,J) ',I,J,B(I,J)
 510        CONTINUE
            IMAX = IMAX - DIM
```

```
 525    CONTINUE
C
C       CALCULATE THE AVERAGE CLASS MEAN VECTOR
C
        SCALE = 1.0 / FLOAT(NCLASS)
        DO 700 I = 1,DIM
            SUM = 0.0
            DO 675 N = 1,NCLASS
                SUM = SUM + MEAN(I,N)
 675        CONTINUE
            AVEMEAN(I) = SUM * SCALE
 700    CONTINUE

C
C       SUBTRACT THE AVERAGE CLASS MEAN FROM EACH CLASS MEAN VECTOR
C
        DO 750 I = 1,DIM
            DO 725 N = 1,NCLASS
                MEAN(I,N) = MEAN(I,N) - AVEMEAN(I)
 725        CONTINUE
 750    CONTINUE

C
C       FORM THE MEAN SCATTER MATRIX SB
C
        DO 800 K = 1,NCLASS
            DO 775 I = 1,DIM
                DO 760 J = 1,DIM
                    SB(I,J) = SB(I,J) + SCALE * (MEAN(I,K) * MEAN(J,K))
 760            CONTINUE
 775        CONTINUE
 800    CONTINUE

C
C       CALCULATE SBSUB = MATRIX SB * MATRIX B
C
        DO 900 I = 1,DIM
            DO 875 J = 1,DIM
                SUM = 0.0
                DO 850 K = 1,DIM
                    SUM = SUM + SB(I,K) * B(K,J)
 850            CONTINUE
                SBSUB(I,J) = SUM
 875        CONTINUE
 900    CONTINUE

C
C       CALCULATE SBPRIME = MATRIX B (TRANSPOSED) * MATRIX SBSUB
C
        DO 1000 I = 1,DIM
            DO 975 J = 1,DIM
                SUM = 0.0
                DO 950 K = 1,DIM
                    SUM = SUM + B(K,I) * SBSUB(K,J)
 950            CONTINUE
                SBPRIME(I,J) = SUM
 975        CONTINUE
1000    CONTINUE
C
C       PUT THE LOWER TRIANGLE OF MATRIX SBPRIME INTO ARRAY A (FOR
```

```
C       SYMMETRIC STORAGE MODE)
C
        K = 1
        DO 1050 I = 1,DIM
            DO 1025 J = 1,I
                A(K) = SBPRIME(I,J)
                K = K + 1
1025        CONTINUE
1050    CONTINUE

C
C       CALCULATE THE EIGENVALUES AND EIGENVECTORS OF SBPRIME (IN A)
C
        CALL EIGRS(A,DIM,JOBN,D,Z,IZ,WK,IER)

C       IMAX = DIM * (DIM - 1)
C       DO 1100 J = 1,DIM
C           K = DIM - J + 1
C           WRITE(DEV,*)
C           WRITE(DEV,*) 'EIGENVALUE: ',D(K)
C           WRITE(DEV,*)
C           DO 1075 I = 1,DIM
C               WRITE(DEV,*) 'EIGENVECTOR: ',Z(IMAX + I)
C1075       CONTINUE
C           IMAX = IMAX - DIM
C1100   CONTINUE

C
C       FORM THE TRANSFORMATION MATRIX BY MULTIPLYING THE B MATRIX
C       TIMES THE FIRST EIGENVECTOR
C
C       WRITE(DEV,*) 'THE TRANSFORMATION MATRIX:'
        DO 1150 I = 1,DIM
            DO 1125 N = 1,DIM
                DO 1115 J = 1,DIM
                    TRANS(I,N) = TRANS(I,N) + B(I,J) * Z(J+DIM*(DIM-N))
1115            CONTINUE
1125        CONTINUE
C           WRITE(DEV,*) 'TRANS(1,N)' ,(TRANS(I,N),N=1,DIM)
1150    CONTINUE
C
C       TRANSFORM THE ORIGNAL SAMPLES OF EACH CLASS
C
        DO 1200 K = 1,NCLASS
            DO 1175 J = 1,NSAMP
                DO 1160 N = 1,DIM
                    Y(J,N,K) = 0.0
                    DO 1155 I = 1,DIM
                        Y(J,N,K) = Y(J,N,K) + (TRANS(I,N) * X(I,J,K))
1155                CONTINUE
1160            CONTINUE
1175        CONTINUE
1200    CONTINUE

        DELTA = SECNDS(T1)
        WRITE(DEV,*) 'PREKL EXECUTION TIME IS: ',DELTA
C       WRITE(DEV,*)'THE TRANSFORMED SAMPLES:'
C       WRITE(DEV,1250) (((Y(J,K,N),K=1,DIM),J=1,NSAMP),N=1,NCLASS)
C1250   FORMAT(1X,<TRDIM>F15.6)

        RETURN
        END
```

```
C*************************************************************************
C
C       FILE CLASSIFY.FOR                LAST REVISION JULY 24,1985
C                                        AUTHOR RUTH D. KENNETT
C
C       THIS FILE CONTAINS:
C
C                          PROGRAM CLASSIFY
C                               CLSHOLE
C                               CLSEDGE
C                               CLSOBJ
C                               OUTCONFUS
C
C       TO LINK:
C
C       LINK CLASSIFY,COLLECT,WORK1,RAD1,PARAM,REGIS,SYS$LIBRARY:VIDEOLIB
C
C*************************************************************************
C                       PROGRAM CLASSIFY
C*************************************************************************
C
C       THIS PROGRAM CLASSIFYS THE OBJECTS
C
        INTEGER DEV,IN,OUT,STEP,INRAD,M
        INTEGER DIM,TRDIM,NSAMP,NCLASS
        INTEGER TRN,TST,ODAT
        INTEGER OBJECT,OBJCT,OBJDAT(0:10,20),NUMHOLE
        INTEGER BOUND(10,10),BNDINT
        INTEGER XPIX(2500,10),YPIX(2500,10)
        INTEGER CLASS(20),EDGECLASS(20)
        INTEGER BNDCT,EDGE,OBJ,PLACE
        INTEGER DECIS,CONFUS(20,20)
        INTEGER GOOD(20),BAD(20),COUNT
        INTEGER TOTAL,GRDTOTAL,TOTGOOD
        REAL    ORIENT(10),LIST(2,10,10)
        REAL    CLDIS(10)
        REAL    TRANS(11,11),Y(11,20)
        REAL    TA(11,20),T(25,11,20)
        REAL    RAD(-2:250,10),THETA(11,10),ALL(10,11,10)
        REAL    DIS,CLSDIS,MINDIS,MAXDIS
        REAL    PERCENT
        CHARACTER INST,SHAPE

        COMMON  DEV,IN,OUT,STEP,INRAD,M

        DATA    MAXDIS/1.7E37/
        DATA    IN,OUT/5,5/
        DATA    CONFUS/400*0/
        DATA    GOOD,BAD/20*0,20*0/

        TRN = 1
        TST = 2
        ODAT = 3
        BNDINT = 1
        OUTEDG = 7
        OPEN(UNIT=TRN, FILE='TRAIN.DAT', STATUS='OLD')
        OPEN(UNIT=TST, FILE='TEST.DAT',STATUS='NEW')
        OPEN(UNIT=ODAT,FILE='OBJECT.DAT',STATUS='OLD')
        CALL SETUP
        CALL THRESH
```

```
      WRITE(OUT,10)
 10   FORMAT(1X,'INPUT DIMENSION OF TRANSFORMED SAMPLES: ',$)
      READ(IN,15) TRDIM
 15   FORMAT(I1)
      WRITE(OUT,20)
 20   FORMAT(1X,'INPUT THE SHAPE CATEGORY: ',$)
      READ(IN,25) SHAPE
 25   FORMAT(A1)
C
C     READ IN THE OBJECT DATA
C

      READ(ODAT,*) OBJCT

      DO 100 N = 1,OBJCT
         READ(ODAT,*)OBJ,NUMHOLE,(OBJDAT(K,OBJ),K=1,NUMHOLE+1)
         OBJDAT(0,OBJ) = NUMHOLE
100   CONTINUE
C
C     READ IN THE DIMENSION OF THE STORED DATA AND CHECK FOR VALIDITY
C

      READ(TRN,*) DIM
      M = DIM - 1

C     READ IN THE STORED DATA: THE TRANSFORMATION MATRIX AND
C     THE TRANSFORMED SAMPLES OF THE CLASSES
C
      DO 110 I = 1,DIM
         READ(TRN,*) (TRANS(I,J),J=1,DIM)
110   CONTINUE

      READ(TRN,*) NSAMP,NCLASS

      DO 160 K = 1,NCLASS
         READ(TRN,*) (TA(J,K),J=1,DIM)
160   CONTINUE

      WRITE(TST,*) '              TEST SET DATA: '
      WRITE(TST,*)
      WRITE(TST,162) (I,I=1,DIM-1)
162   FORMAT(1X,'OBJECT HOLES ROTATE(DEG) MAXRAD',3X,
C    *  <DIM-2>('M',I1,7X),'M',I1,3X,'ALPHA/SQRT(BETA)'// )
     *  'M',I1,3X,'ALPHA/SQRT(BETA)'//)

      WRITE(DEV,*) '              CLASSIFICATION DATA:'
      WRITE(DEV,*)
      WRITE(DEV,170) M,TRDIM,SHAPE
170   FORMAT(1X,'AR MODEL ORDER: ',I2,/' DIMENSION OF TRANSFORMED ',
     *  'SAMPLES: ',I3,/' SHAPES CATEGORY: ',A1,//)

      WRITE(DEV,175) (J,J=1,TRDIM), (J,J=1,NCLASS)

175   FORMAT(1X,'OBJECT EDGE',3X,<TRDIM>('T',I2,7X),2X,
     *  <NCLASS>('D',I1,8X), 'EDGE  OBJECT'/)

      COUNT = 0
200   CALL VIEW
      WRITE(OUT,210) COUNT
```

```
 210     FORMAT(1X,I3,' CLASSIFICATIONS, DONE YET? (TYPE Y FOR YES): ',$)
         READ(IN,220) INST
 220     FORMAT(A1)
         IF(INST .EQ. 'Y') GOTO 900

         WRITE(OUT,230)
 230     FORMAT(1X,'INPUT THE OBJECT NUMBER: ',$)
         READ(IN,240) OBJECT
 240     FORMAT(I2)
C
C        SCAN THE FRAME BUFFER FOR THE EDGES OF THE UNCLASSIFIED OBJECT
C
         T1 = SECNDS(0.00)
         DO 250 I = 1,10
             CLASS(I) = 0.0
 250     CONTINUE

         T1 = SECNDS(0.00)

         CALL SNAP
         CALL SCAN(BNDCT,BOUND,XPIX,YPIX)
         IF(BNDCT .EQ. 0) GOTO 999
         CALL INSIDE(BNDCT,BOUND)
C
C        CLASSIFY USING HOLE INFORMATION (IF POSSIBLE)
C
         CALL CLSHOLE(OBJECT,BNDCT,BOUND,CLASS,OBJDAT,OBJCT)
C
C        LIST THE INNER EDGES IN ORDER OF THE DISTANCE BETWEEN THE INNER EDGE
C        CENTROID AND THE OUTER EDGE CENTROID (LARGEST DISTANCE IS FIRST IN
C        LIST)
C
         CALL ORDER(BNDCT,BOUND,LIST)
C
C        CLASSIFY THE REST OF THE OBJECT USING THE AUTOREGRESIVE PARAMETERS
C        OF THE OBJECT EDGES
C
         DO 800 OBJ = 1,BNDCT
             IF(CLASS(OBJ) .EQ. 0) GOTO 400
C            WRITE(TST,300) OBJECT,BOUND(8,OBJ),ORIENT(OBJ),
C     *      INT(RAD(-1,OBJ))
C 300       FORMAT(1X,2(I3,3X),F8.2,3X,I5)
             GOTO 700
 400         PLACE = 1
 500         IF(LIST(BNDINT,PLACE,OBJ) .NE. 0.0) GOTO 550
             IF(PLACE .GT. 1) GOTO 600
             GOTO 800
 550         EDGE = INT(LIST(BNDINT,PLACE,OBJ))
C
C            CALCULATE THE RADIUS VECTOR LENGTHS
C
 560         CALL GETRAD(XPIX(1,EDGE),YPIX(1,EDGE),EDGE,BOUND,RAD)
C
C            CALCULATE THE AUTOREGRESSIVE PARAMETERS
C
             CALL PARAM(EDGE,RAD,THETA,ALL)

             WRITE(TST,575) OBJECT,BOUND(8,EDGE),ORIENT(EDGE),
     *       INT(RAD(-1,EDGE)),(THETA(I,EDGE),I=1,DIM)
 575         FORMAT(1X,2(I3,3X),F8.2,3X,I5,3X,(DIM-1)F9.5,F11.5)
```

```
C            CLASSIFY THE EDGE USING THE TRANSFORMED PARAMETERS
C
             CALL CLSEDGE(OBJECT,EDGE,THETA,TRANS,TA,TRDIM,NCLASS,DECIS)
             EDGECLASS(PLACE) = DECIS
             PLACE = PLACE + 1
             GOTO 500
C
C            CLASSIFY THE OBJECT BASED ON ALL THE EDGE CLASSIFICATIONS
C
 600         CALL CLSOBJ(OBJECT,OBJ,EDGECLASS,OBJDAT,OBJCT,CLASS,
     *       BOUND(8,OBJ),TRDIM,NCLASS)

 700         DELTA = SECNDS(T1)
C            WRITE(DEV,725) OBJECT,CLASS(OBJ),DELTA
C 725        FORMAT(1X,'OBJECT ',I2,' IS: ',I2,/' CLASSIFY TIME IS: ',
C     *       F10.6/)
C
C            CALCULATE THE ORIENTATION OF THE EDGES
C
             CALL ROTATE(EDGE,BOUND,ORIENT)
             CALL OUTPUT(EDGE,BOUND,RAD,ORIENT)

             COUNT = COUNT + 1
             CONFUS(OBJECT,CLASS(OBJ)) = CONFUS(OBJECT,CLASS(OBJ)) + 1
             IF(CLASS(OBJ) .EQ.  OBJECT) GOTO 750
             BAD(OBJECT) = BAD(OBJECT) + 1
             GOTO 800
 750         GOOD(OBJECT) = GOOD(OBJECT) + 1
 800     CONTINUE

         GOTO 200

 900     CALL OUTCONFUS(CONFUS,OBJCT)

         WRITE(DEV,*)
         WRITE(DEV,*) 'CLASS   PERCENT CORRECT'
         GRDTOTAL = 0.0
         DO 950 K = 1,OBJCT
             TOTAL = GOOD(K) + BAD(K)
             IF(TOTAL .EQ. 0) GOTO 950
             PERCENT = 100.0 * FLOAT(GOOD(K)) / FLOAT(TOTAL)
             WRITE(DEV,940) K,PERCENT
 940         FORMAT(1X,I3,5X,F10.5)
             GRDTOTAL = GRDTOTAL + PERCENT
 950     CONTINUE

         PERCENT = GRDTOTAL / FLOAT(OBJCT)
         WRITE(DEV,960) PERCENT
 960     FORMAT(1X,'OVERALL PERCENT CORRECT: ',F10.5)
         GOTO 1000

 999     WRITE(OUT,*) 'NO OBJECTS FOUND'

1000     CLOSE(UNIT=ODAT)
         CLOSE(UNIT=TRN)
         CLOSE(UNIT=TST)
         STOP
         END
```

```
C-----------------------------------------------------------------------
C                                CLSHOLE
C-----------------------------------------------------------------------

      SUBROUTINE CLSHOLE(OBJECT,BNDCT,BOUND,CLASS,OBJDAT,OBJCT)

C
C     THIS SUBROUTINE CLASSIFIES THE OBJECT ON THE BASIS OF THE NUMBER
C     OF HOLES IT HAS.
C

      INTEGER DEV,IN,OUT
      INTEGER BNDCT,BOUND(10,10),CLASS(10)
      INTEGER BND,OBJ
      INTEGER MAXHOLE,NUMHOLE,NHOLE(2,10)
      INTEGER OBJECT,OBJCT,OBJDAT(0:10,20)
      COMMON DEV,IN,OUT

      DATA    NHOLE/20*0/

      BND = 1
      OBJ = 2
      MAXHOLE = 10

      DO 100 I = 1,10
          DO 50 J = 1,2
              NHOLE(J,I) = 0
       50     CONTINUE
      100 CONTINUE

      DO 400 N = 1,BNDCT
          DO 300 K = 1,OBJCT
              IF(BOUND(8,N) .NE. OBJDAT(0,K)) GOTO 300
              NUMHOLE = BOUND(8,N)
              IF(NHOLE(BND,NUMHOLE) .EQ. 0) GOTO 250
              NHOLE(BND,NUMHOLE) = -1
              GOTO 300
      250     NHOLE(BND,NUMHOLE) = N
              NHOLE(OBJ,NUMHOLE) = K
      300     CONTINUE
      400 CONTINUE

      DO 500 NUMHOLE = 1,MAXHOLE
          IF(NHOLE(BND,NUMHOLE) .LE. 0) GOTO 500
          CLASS(NHOLE(BND,NUMHOLE)) = NHOLE(OBJ,NUMHOLE)
C         WRITE(DEV,450) OBJECT,NHOLE(BND,NUMHOLE),NUMHOLE,
C    *    NHOLE(OBJ,NUMHOLE)
C450      FORMAT(1X,2(I3,3X),'CLASSIFICATION BASED ON: ',I1,' HOLES:',I3)
      500 CONTINUE

      CLOSE(UNIT=LUN)

      RETURN
      END

C-----------------------------------------------------------------------
C                                CLSEDGE
C-----------------------------------------------------------------------

      SUBROUTINE CLSEDGE(OBJECT,EDGE,THETA,TRANS,TA,TRDIM,NCLASS,DECIS)
```

```
C
C       THIS SUBROUTINE CLASSIFIES AN EDGE OF THE OBJECT.  THE CLASSIFICATION
C       IS BASED ON THE SMALLEST DISTANCE FROM THE TRANSFORMED PARAMETERS OF
C       THE UNCLASSIFIED EDGE TO THE AVERAGE OF THE TRANSFORMED SAMPLES OF THE
C       TRAINING SET OF EDGES.
C
        INTEGER DEV,IN,OUT,STEP,INRAD,M
        INTEGER EDGE,TRDIM,NCLASS,DECIS
        INTEGER OBJECT,MINCLS
        REAL    MAXDIS,MINDIS
        REAL    THETA(11,10),TA(11,20)
        REAL    TRANS(11,11),Y(11,20)
        REAL    CLSDIS(10)

        COMMON  DEV,IN,OUT,STEP,INRAD,M

        DATA    MAXDIS/1.7E37/

        N = EDGE
        DIM = M + 1
C
C       TRANSFORM THE AR PARAMETERS OF THE UNCLASSIFIED EDGE
C
        DO 200 J = 1,TRDIM
            Y(J,N) = 0.0
            DO 150 I = 1,DIM
                Y(J,N) = Y(J,N) + (TRANS(I,J) * THETA(I,N))
 150        CONTINUE
 200    CONTINUE
C
C       FIND THE SMALLEST DISTANCE TO A CLASS
C
        MINDIS = MAXDIS
        MINCLS = 0
        DO 400 K = 1,NCLASS
            CLSDIS(K) = 0.0
            DO 300 J = 1,TRDIM
                CLSDIS(K) = CLSDIS(K) + (Y(J,N) - TA(J,K))**2
300         CONTINUE
            IF(CLSDIS(K) .GT. MINDIS) GOTO 400
            MINDIS = CLSDIS(K)
            MINCLS = K
 400    CONTINUE

        DECIS = MINCLS

        WRITE(DEV,450) OBJECT,EDGE,(Y(J,N),J=1,TRDIM),(CLSDIS(K),K=1,NCLASS),
     *  DECIS
 450    FORMAT(1X,2(I3,3X),<TRDIM>F12.7,<NCLASS>F12.7,8X,I3)

        RETURN
        END

C-------------------------------------------------------------------------
C                                CLSOBJ
C-------------------------------------------------------------------------

        SUBROUTINE CLSOBJ(OBJECT,OBJ,EDGECLASS,OBJDAT,OBJCT,OBJCLASS,THOLE,
     *  TRDIM,NCLASS)
C
```

```
C       THIS SUBROUTINE CLASSIFIES THE OBJECT USING THE EDGE CLASSIFICATIONS
C
        INTEGER DEV,IN,OUT
        INTEGER EDGECLASS(10),OBJDAT(0:10,20)
        INTEGER OBJECT,OBJ,OBJCT,OBJCLASS(10)
        INTEGER COUNT,OBJHOLE,THOLE
        INTEGER TRDIM,NCLASS

        COMMON  DEV,IN,OUT

        N = 1
 100    IF(N .GT. OBJCT) GOTO 600
        OBJHOLE = OBJDAT(0,N)
        IF(THOLE .NE. OBJHOLE) GOTO 300

        COUNT = 0
        DO 200 J = 1,OBJHOLE+1
            IF(EDGECLASS(J) .NE. OBJDAT(J,N)) GOTO 200
            COUNT = COUNT + 1
 200    CONTINUE

        IF(COUNT .EQ. OBJHOLE+1) GOTO 500
 300    N = N + 1
        GOTO 100

 500    OBJCLASS(OBJ) = N
        GOTO 1000

 600    OBJCLASS(OBJ) = 0

 1000   CONTINUE
C 1000  WRITE(DEV,1010) OBJCLASS(OBJ)
C 1010  FORMAT(1X,12X,<TRDIM>(8X),<NCLASS>(10X),14X,I3)
        RETURN
        END

C-------------------------------------------------------------------------
C                                OUTCONFUS
C-------------------------------------------------------------------------

        SUBROUTINE OUTCONFUS(CONFUS,NCLASS)
C
C       THIS SUBRUTINE WRITES OUT THE CONFUSION MATRIX OF THE CLASSIFI-
C       CATION DATA
C
        INTEGER CONFUS(20,20),NCLASS
        INTEGER DEV

        COMMON  DEV

        WRITE(DEV,100)
 100    FORMAT(1X,'THE CONFUSION MATRIX OF THE CLASSIFICATION DATA'//)

        WRITE(DEV,150) (I,I=1,NCLASS)
 150    FORMAT(1X,6X,<NCLASS>I5,/7X,<NCLASS*5>('-'))

        DO 400 I = 1,NCLASS
            WRITE(DEV,200) I,(CONFUS(I,J),J=1,NCLASS)
 200        FORMAT(1X,I5,'|',<NCLASS>I5)

 400    CONTINUE

        RETURN
        END
```

```
C*************************************************************************************
C
C       FILE COLLECT.FOR   LAST REVISION JUNE 10, 1985
C
C
C       SUBROUTINES CONTAINED IN THIS FILE ARE:
C
C                       INSIDE
C                       ROTATE
C                       OUTPUT
C                       ORDER
C
C*************************************************************************************
C-------------------------------------------------------------------------------------
C                                 INSIDE
C-------------------------------------------------------------------------------------

        SUBROUTINE INSIDE(BNDCT,BOUND)
C
C       THIS SUBROUTINE FINDS OUT WHICH BOUNDARIES ARE INSIDE OTHER BOUNDARIES
C
        INTEGER BNDCT,BOUND(10,10),BNDINT
        INTEGER XR,XL,XPREV,RPIXEL,Z
        INTEGER DEV,IN,OUT,STEP,INRAD
        INTEGER OBJ,COUNT
        INTEGER YLINE,FIRSTX,LASTX,STARTX,XINC,BUFF(0:511)
        LOGICAL EDGE,RBND(20),LBND(20)

        COMMON  DEV,IN,OUT,STEP,INRAD

        DO 40 I = 1,BNDCT
            BOUND(7,I) = 0
            BOUND(8,I) = 0
 40     CONTINUE

        FIRSTX = 0
        LASTX = 511
        XINC = 1
        EDGE = .TRUE.
C
C       LOOP FOR EACH EDGE
C
        DO 600 N = 1,BNDCT
            BNDINT = BOUND(1,N)
            STARTX = BOUND(2,N)
            YLINE = BOUND(3,N)

            DO 100 I = 1,BNDCT
                RBND(I) = .FALSE.
                LBND(I) = .FALSE.
 100        CONTINUE
C
C           SEARCH FOR EDGES ON THE RIGHT
C
            XPREV = 0
            CALL RSCAN(YLINE,FIRSTX,LASTX,XINC,BUFF)

            DO 200 XR = STARTX,LASTX
                Z = BUFF(XR)
                IF((Z.EQ.0).OR.(Z.EQ.255).OR.(Z.EQ.BNDINT)) GOTO 200
```

```
                IF(XR .NE. (XPREV + 1)) GOTO 150
                XPREV = XR
                GOTO 200
 150            RBND(Z) = RBND(Z) .NEQV. EDGE
                XPREV = XR
 200        CONTINUE
C
C           SEARCH FOR EDGES ON THE LEFT
C
            XPREV = 0
            DO 300 XL = STARTX,FIRSTX,-1
                Z = BUFF(XL)
                IF((Z.EQ.0).OR.(Z.EQ.255).OR.(Z.EQ.BNDINT)) GOTO 300
                IF(XL .NE. (XPREV - 1)) GOTO 250
                XPREV = XL
                GOTO 300
 250            LBND(Z) = LBND(Z) .NEQV. EDGE
                XPREV = XL
 300        CONTINUE
C
C           COMPARE THE RIGHT AND LEFT TALLIES
C
            COUNT = 0
            DO 400 I = 1,BNDCT
                IF(.NOT.(RBND(I) .AND. LBND(I))) GOTO 400
                COUNT = COUNT + 1
                OBJ = I
                BOUND(7,N) = I                  !FOUND AN OUTER EDGE
 400        CONTINUE
            IF(COUNT .EQ. 1) GOTO 450
            BOUND(7,N) = 0
            GOTO 600
 450        BOUND(8,OBJ) = BOUND(8,OBJ) + 1     !COUNT HOLES INSIDE OUTER EDGE
 600    CONTINUE

        RETURN
        END

C-------------------------------------------------------------------------------
C                                ROTATE
C-------------------------------------------------------------------------------

        SUBROUTINE ROTATE(BNDCT,BOUND,ORIENT)
C
C       THIS SUBROUTINE CALCULATES THE ROTATION OF THE MAXIMUM RADIUS VECTOR
C       (IN DEGREES) IN REFERENCE TO THE STANDARD CARTESION COORDINATE SYSTEM.
C
        INTEGER BNDCT,BOUND(10,10)
        INTEGER XC,YC,XMAX,YMAX
        REAL    ORIENT(10)
        REAL    XDIF,YDIF,THETA

        XC = 4
        YC = 5
        XMAX = 9
        YMAX = 10

        DO 500 I = 1,BNDCT
            XDIF = FLOAT(BOUND(XMAX,I) - BOUND(XC,I))
            YDIF = -FLOAT(BOUND(YMAX,I) - BOUND(YC,I))
```

```
            IF(XDIF .NE. 0.0) GOTO 400
            IF(YDIF .GT. 0.0) GOTO 300
            THETA = 90.0
            GOTO 490
 300        THETA = 270.0
            GOTO 490
 400        THETA = ATAND(YDIF/XDIF)
            IF(XDIF .LT. 0.0) THETA = THETA + 180.0
            IF(THETA .LT. 0.0) THETA = THETA + 360.0
 490        ORIENT(I) = THETA
 500    CONTINUE

        RETURN
        END


C-------------------------------------------------------------------------
C                                OUTPUT
C-------------------------------------------------------------------------

        SUBROUTINE OUTPUT(BNDCT,BOUND,RAD,ORIENT)
C
C       THIS SUBROUTINE PRODUCES THE OUTPUT OF THE OBJECT DATA
C
        INTEGER BNDCT,BOUND(10,10)
        INTEGER DEV
        REAL    RAD(-2:300,10),ORIENT(10)

        COMMON DEV

        WRITE(DEV,50)
 50     FORMAT(1X,'EDGE',6X,'START',8X,'CENTER',6X,'NUMPIX',2X,
     *          'INSIDE',2X,' HOLES','   ROT (DEG)','  # RAD VEC',
     *          '   MAX RAD VEC (PIX)'/)

        DO 800 N = 1,BNDCT
            WRITE(DEV,750) (BOUND(I,N),I=1,8),ORIENT(N),RAD(-2,N),
     *      RAD(-1,N)
 750        FORMAT(1X,I3,5X,'(',I3,',',I3,')',5X,'(',I3,',',
     *          I3,')',4X,I4,2(5X,I3),6X,F5.1,2(4X,F8.2))
 800    CONTINUE

C       WRITE(DEV,810)(N,N = 1,BNDCT)
C 810   FORMAT(//,<BNDCT>(6X,'EDGE')/<BNDCT>(7X,'(',I1,')'),/)
C       WRITE(DEV,820)
C 820   FORMAT(/,' NUMBER OF RADIUS VECTORS:',/)
C       WRITE(DEV,825) (RAD(-2,N),N=1,BNDCT)
C 825   FORMAT(1X,3X,8F10.4)
C       WRITE(DEV,830)
C 830   FORMAT(/,' MAXIMUM RADIUS VECTOR: ',/)
C       WRITE(DEV,835) (RAD(-1,N),N=1,BNDCT)
C 835   FORMAT(1X,3X,8F10.4)
C
C       WRITE(DEV,850)
C 850   FORMAT(/,' RADIUS VECTOR LENGTHS:'/)
C
C       DO 950 I = 1,INRAD+1
C           WRITE(DEV,920) I,(RAD(I,N),N=1,BNDCT)
C 920       FORMAT(1X,I3,8F10.4)
C 950   CONTINUE
```

```
 310        IF(LIST(DIS,PLACE+1,OBJ) .EQ. 0.0) GOTO 320
            PLACE = PLACE + 1
            GOTO 310

 320        LIST(BNDINT,PLACE+1,OBJ) = LIST(BNDINT,PLACE,OBJ)
            LIST(DIS,PLACE+1,OBJ) = LIST(DIS,PLACE,OBJ)
            IF(PLACE .LE. MARK) GOTO 340
            PLACE = PLACE - 1
            GOTO 320

 340        LIST(BNDINT,PLACE,OBJ) = FLOAT(N)
            LIST(DIS,PLACE,OBJ) = DIST

 500    CONTINUE

C       WRITE(DEV,550)
C 550   FORMAT(1X,///' OUTER EDGE    INNER EDGE     ORDER         DISTANCE' /)

        DO 800 OBJ = 1,BNDCT
            PLACE = 1
 720        IF(LIST(BNDINT,PLACE,OBJ) .EQ. 0.0) GOTO 800
            INEDGE = INT(LIST(BNDINT,PLACE,OBJ))
            IF(INEDGE .EQ. OBJ) GOTO 730
            ORD = PLACE
            GOTO 735
 730        INEDGE = 0
            ORD = 0
 735        CONTINUE
C           WRITE(DEV,740) OBJ,INEDGE,ORD,LIST(DIS,PLACE,OBJ)
C 740      FORMAT(1X,I6,9X,I3,11X,I2,7X,E10.4)
            PLACE = PLACE + 1
            GOTO 720
 800    CONTINUE

        RETURN
        END
```

END

10-86

DTIC